Pontifícia Universidade Católica
do Rio de Janeiro

Robinson Callou de Moura Brasil Filho

Arguing NP = PSPACE: On the Coverage and Soundness of the Horizontal Compression Algorithm

Tese de Doutorado

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática.

Advisor: Prof. Edward Hermann Haeusler

Rio de Janeiro
April 2024

Pontifícia Universidade Católica
do Rio de Janeiro

Robinson Callou de Moura Brasil Filho

Arguing NP = PSPACE: On the Coverage and Soundness of the Horizontal Compression Algorithm

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática. Approved by the Examination Committee:

Prof. Edward Hermann Haeusler
Advisor
Departamento de Informática – PUC-Rio

Prof. Alex de Vasconcellos Garcia
IME

Prof. Alexandre Rademaker
Fundação Getúlio Vargas – Matriz

Prof. Bernardo Pinto de Alkmim
Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio

Prof. Jefferson de Barros Santos
FGV

Prof. Mario Roberto Folhadela Benevides
UFF

Prof. Mauricio Ayala Rincon
UnB

Rio de Janeiro, April 30th, 2024

Robinson Callou de Moura Brasil Filho

Has a Bachelor's Degree in Computer Engineering, class of 2016, from the Military Institute of Engineering (IME, Rio de Janeiro, Brazil). Has a MSc in Computer Science from the Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio, Rio de Janeiro, Brazil), specialising in Formal Verification, Proof Theory and Theory of Computation.

Bibliographic data

Filho, Robinson Callou de Moura Brasil

Arguing NP = PSPACE: On the Coverage and Soundness of the Horizontal Compression Algorithm / Robinson Callou de Moura Brasil Filho; advisor: Edward Hermann Haeusler. – 2024.

57 f. : il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2024.

Inclui bibliografia

1. Informática – Teses. 2. Compressão de Provas. 3. Lógica Minimal Puramente Implicacional. 4. Grafos Acíclicos Dirigidos. 5. Prova Interativa de Teoremas. 6. Complexidade de Algoritmos. I. Haeusler, Edward Hermann. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

I thank my teacher and advisor, Prof. Edward Hermann Haeusler. Thank you for all these years we've been working together, during both my masters and my doctorate. Thank you, once again, for believing I could see this work to fruition, even if, at many points, I myself may have not believed it possible.

I thank all the members of the Examination Committee, Prof. Alex de Vasconcellos Garcia, Prof. Alexandre Rademaker, Prof. Jefferson de Barros Santos, and Prof. Mauricio Ayala Rincon. Thank you for reviewing my work and for the contributions you've made.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Filho, Robinson Callou de Moura Brasil; Haeusler, Edward Hermann (Advisor). **Arguing NP = PSPACE: On the Coverage and Soundness of the Horizontal Compression Algorithm**. Rio de Janeiro, 2024. 57p. Doctorate Thesis – Departamento of Informatics, Pontifícia Universidade Católica do Rio de Janeiro.

This work is an elaboration, with examples, and evolution of the presented Horizontal Compression Algorithm (HC) and its set of Compression Rules. This work argues a proof, done in the Lean Interactive Theorem Prover, that the HC algorithm can obtain a Compressed Derivation, represented by a Directed Acyclic Graph, from any Tree-Like Natural Deduction Derivation in Minimal Purely Implicational Logic. Finally, from the Coverage and Soundness of the HC algorithm, one can argue that NP = PSPACE.

Keywords

Proof Compression; Purely Implicational Minimal Logic; Directed Acyclic Graphs; Interactive Theorem Proving; Algorithmic Complexity.

Resumo

Filho, Robinson Callou de Moura Brasil; Haeusler, Edward Hermann. **Argumentando NP = PSPACE: Sobre a Cobertura e Corretude do Algoritmo de Compressão Horizontal**. Rio de Janeiro, 2024. 57p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho é uma elaboração, com exemplos, e evolução do Algoritmo de Compressão Horizontal (HC) apresentado e seu Conjunto de Regras de Compressão. Este trabalho apresenta uma prova, feita no Provador Interativo de Teoremas Lean, de que o algoritmo HC pode obter uma Derivação Comprimida, representada por um Grafo Acíclico Dirigido, a partir de qualquer Derivação Tipo-Árvore em Dedução Natural para a Lógica Minimal Puramente Implicacional. Finalmente, a partir da Cobertura e Corretude do algoritmo HC, pode-se argumentar que NP = PSPACE.

Palavras-chave

Compressão de Provas; Lógica Minimal Puramente Implicacional; Grafos Acíclicos Dirigidos; Prova Interativa de Teoremas; Complexidade de Algoritmos.

Table of contents

List of figures

Macte virtute, sic itur ad astra.

Vergilius, *AENEIS, IX, 641.*

1
Introduction

This document presents a doctoral thesis of the Department of Informatics at PUC-Rio about the representation and size of proofs in natural deduction, proof compression in natural deduction, formal verification, and the use of interactive theorem provers. The main result of this thesis is the proof/formalization, done in Lean, that a valid **D**ag-**L**ike **D**erivability **S**tructure (**DLDS**) is obtained by compressing any natural deduction derivation of the purely implicational minimal logic ($\mathbf{M}_{\supset}$) via an application of the **M**oving **U**pwards **E**dges (**MUE**) phase of the Horizontal Compression Algorithm (**HC**), made avaible in [1].

Proof compression has many significant uses, meaning that an algorithm like **HC** would have many practical applications. Many problems require the formal verification of the proofs of their solution. In computer science, for example, formal verification techniques can be used to guarantee the correctness of algorithms, systems, and protocols. Compressing the size of these proofs reduces makes their verification faster and more efficient. Large proofs are also computationally expensive to store and manipulate. By compressing a proof, one reduces the space required to store it. Computation on smaller sized proofs, such as proof-checking or the transmission of the proof as a file, are also faster than on bigger ones. This would be specially relevant in resource constrained environments. In essence, proof compression allows for a more sensible use of resources, without compromising the integrity of verification. It could even be what makes verification at all feasible in the case of particularly complex or huge proofs.

The formalization in Lean, though not a 1-to-1 translation of the mathematical proofs shown in [2], was indeed based both on the definition of **HC** and on the proofs of its properties as shown in [2]. The proof shows, both as a lemma and as a corollary, that not only the final, fully-compressed **DLDS**, but also the initial tree-like derivation and all other intermediary graph-like structures produced by **HC** are valid **DLDS**. This work also explains how **HC** has changed and improved over time, much in part due to insights taken from the effort of formalisation. This sort of moving goalpost approach to the development of the algorithm **HC** in parallel with the formalisation of its properties

has so far been indeed beneficial, though sometimes laborious. These developments should also be regarded as one of this work's contributions.

The size $|\Pi|$ of a formal proof/derivation Π is considered to be the number of occurrences of letters in the word obtained by linearizing its representation[3]. Proofs/derivations in natural deduction are usually represented using trees (proof-trees), therefore proof compression for proofs/derivations in natural deduction is the same as compressing the information on these proof-trees. An idea to compress these proof-trees is to first view them as proof-graphs. One can then manipulate these proof-graphs, by enriching their representation with additional information, in order to prune their redundancy, producing smaller, compressed proofs/derivations, so long as the gains from pruning said redundancies makes up for any added information. Thus, using graph-like structures to represent logical proofs and derivations can provide shorter sized proof objects, in comparison to using tree-like structures as is commonplace in natural deduction.

One way to implement such a compression is via the horizontal collapse of nodes with repeating formulas in the proof tree. The **HC** algorithm, for the horizontal, left-to-right compression procedure that it applies, is based on the collapses of equally labeled nodes in the original tree-like proof/derivation, producing one valid **D**ag-**L**ike **D**erivability **S**tructure (**DLDS**) from another valid **DLDS**, interactively, via a finite set of compression rules, each with its specific conditions.

The compression rules of **HC** can be grouped in many ways. One such grouping of the rules, the most important, in fact, in regards to the scope of this work, is the distinction made between **MUE**-rules, for **M**oving **U**pwards **E**dges, and **MDE**-rules, for **M**oving **D**own **E**dges[2]. All valid **DLDS**, like any graph, constitute of a set of nodes and a set of edges, which in the case of a DLDS can be separated into deduction-edges, entirely inherited from the original proof-tree, and ancestral-paths (or ancestral-edges), created during the collapse to maintain derivation integrity and entirely missing from the original proof-tree. The **MUE**-rules are those rules used to either move up the ancestral-paths when collapsing nodes that are the end-point of these edges or create new ancestral-paths when the underlying structure requires so after the collapsing of a node. Most of the rules of **HC** are **MUE**-rules and most of the compression rate is due to these **MUE**-rules. The **MDE**-rules are those rules used solely when the collapse involves a hypothesis-node (or top-node; or top-formula), with the added effect of moving down, and sometimes dismissing, ancestral-paths. These **MDE**-rules are invoked only at the end of the collapsing of an entire sub-proof.

Another way that the compression rules of **HC** can be grouped is the concept of the type hierarchy of **HC**, briefly defined as follows, using a nomenclature akin to that of the formalisation:

- `Type-0 Elimination`, 6.2 which validates if a given neighborhood of the **DLDS** represents a non-collapsed node that is the conclusion of an application of $\supset$-Elimination, as it was in the original tree-like proof, with a non-collapsed parent-node or incoming ancestral-paths;
- `Type-0 Introduction`, 6.3 which validates if a given neighborhood of the **DLDS** represents a non-collapsed node that is the conclusion of an application of $\supset$-Introduction, as it was in the original tree-like proof, with a non-collapsed parent-node or incoming ancestral-paths;
- `Type-0 Hypothesis`, 6.4 which validates if a given neighborhood of the **DLDS** represents a non-collapsed node that is a hypothesis, as it was in the original tree-like proof, with a non-collapsed parent-node or incoming ancestral-paths;

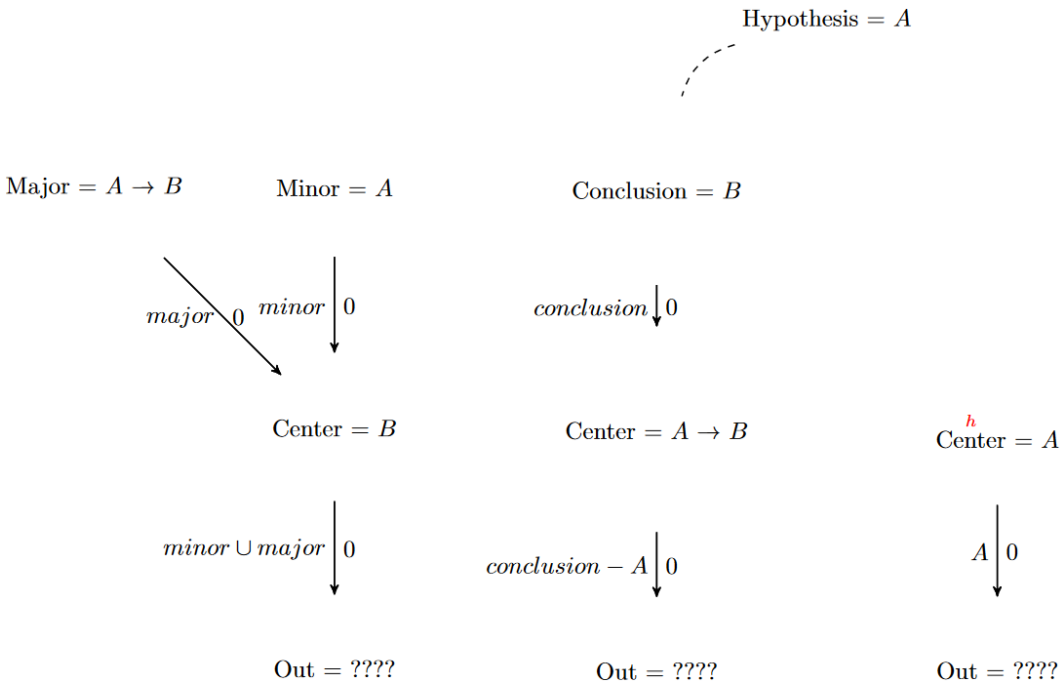

Figure 1.1: Representation of the Type-0 Neighborhoods, from left-to-right: Type-0 Elimination, Type-0 Introduction and Type-0 Hypothesis.

- `Type-1 Collapse`, 6.6 which validates if a given neighborhood of the **DLDS** represents a collapsed node resulting from the collapse: of a type-0 and a type-0; or a type-1 and a type-0 nodes;

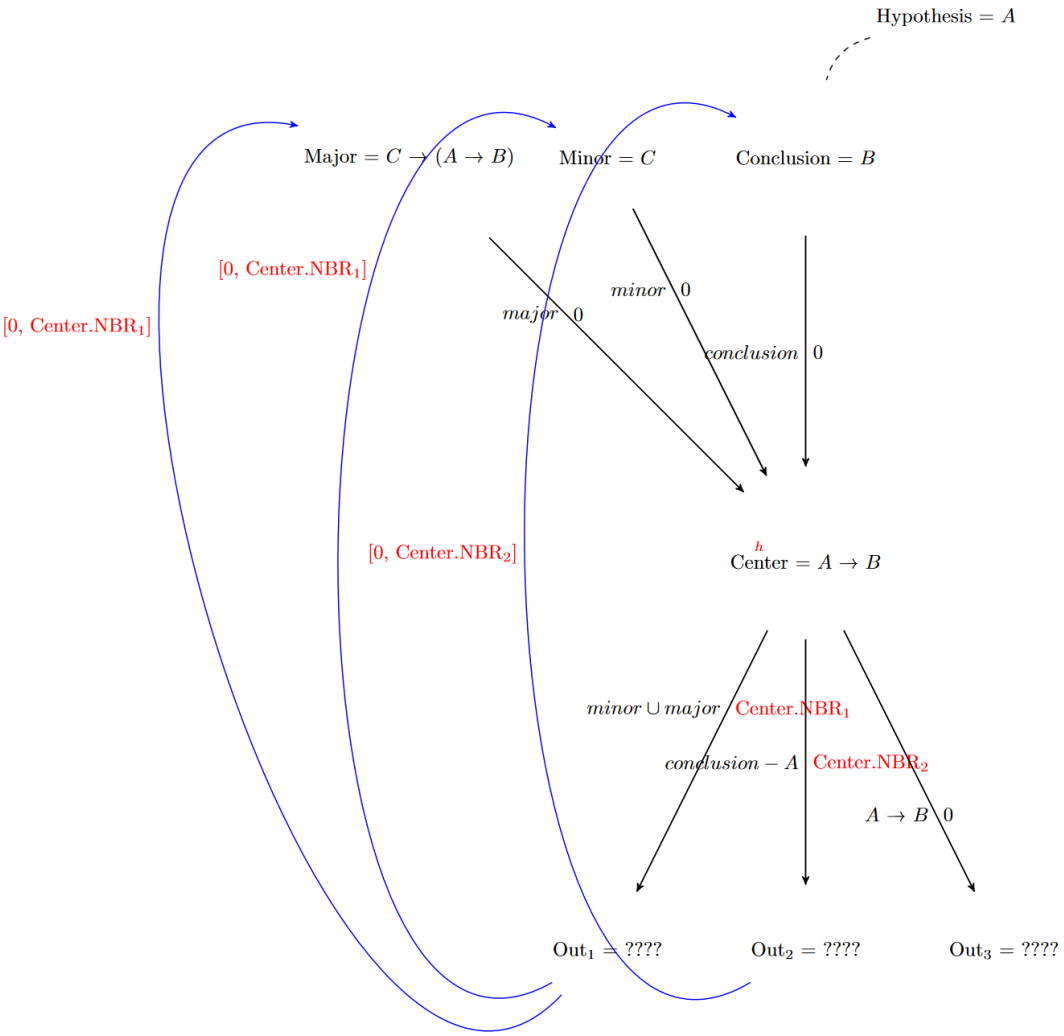

Figure 1.2: Representation of a generic Type-1 Collapse Neighborhood.

➡ `Type-2 Elimination`, 6.7 which validates if a given neighborhood of the **DLDS** represents a non-collapsed node that is the conclusion of an application of $\supset$-Elimination, but not as it was in the original tree-like proof, with a collapsed parent-node and a single incoming ancestral-path;

➡ `Type-2 Introduction`, 6.8 which validates if a given neighborhood of the **DLDS** represents a non-collapsed node that is the conclusion of an application of $\supset$-Introduction, but not as it was in the original tree-like proof, with a collapsed parent-node and a single incoming ancestral-path;

➡ `Type-2 Hypothesis`, 6.9 which validates if a given neighborhood of the **DLDS** represents a non-collapsed node that is a hypothesis, but not as it was in the original tree-like proof, with a collapsed parent-node and a single incoming ancestral-path;

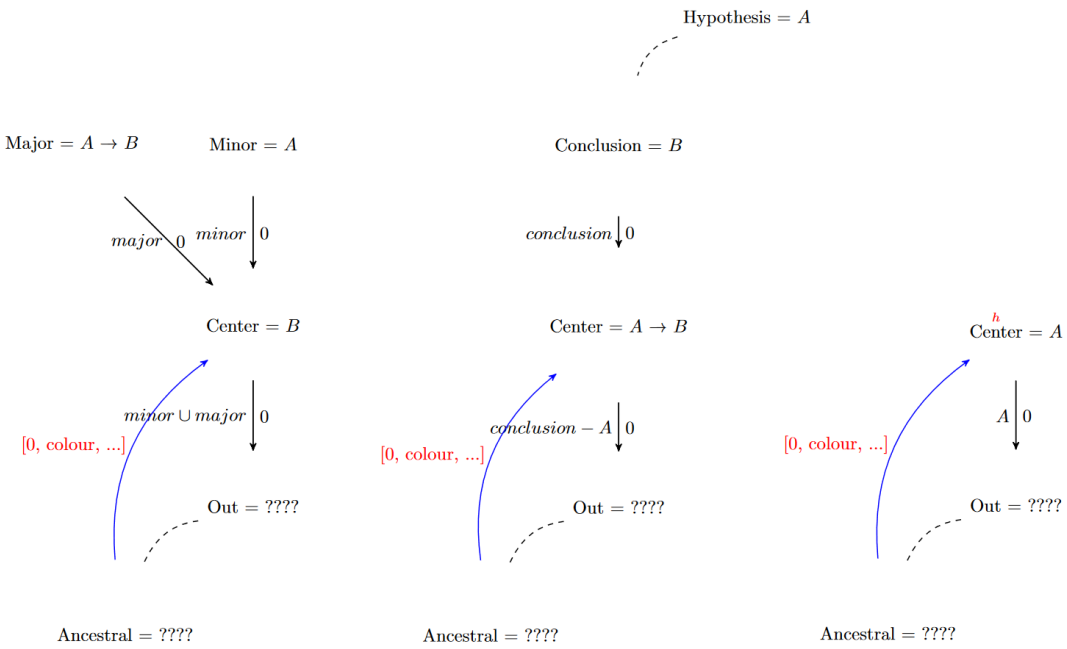

Figure 1.3: Representation of the Type-2 Neighborhoods, from left-to-right: Type-2 Elimination, Type-2 Introduction and Type-2 Hypothesis.

➡ `Type-3 Collapse`, 6.11 which validates if a given neighborhood of the **DLDS** represents a collapsed node resulting from the collapse of: a type-0 and a type-0; or a type-0 and a type-2; or a type-2 and a type-0; or a type-2 and a type-2; or a type-1 and a type-0; or a type-1 and a type-2; or a type-3 and a type-0; or a type-3 and a type-2 nodes.

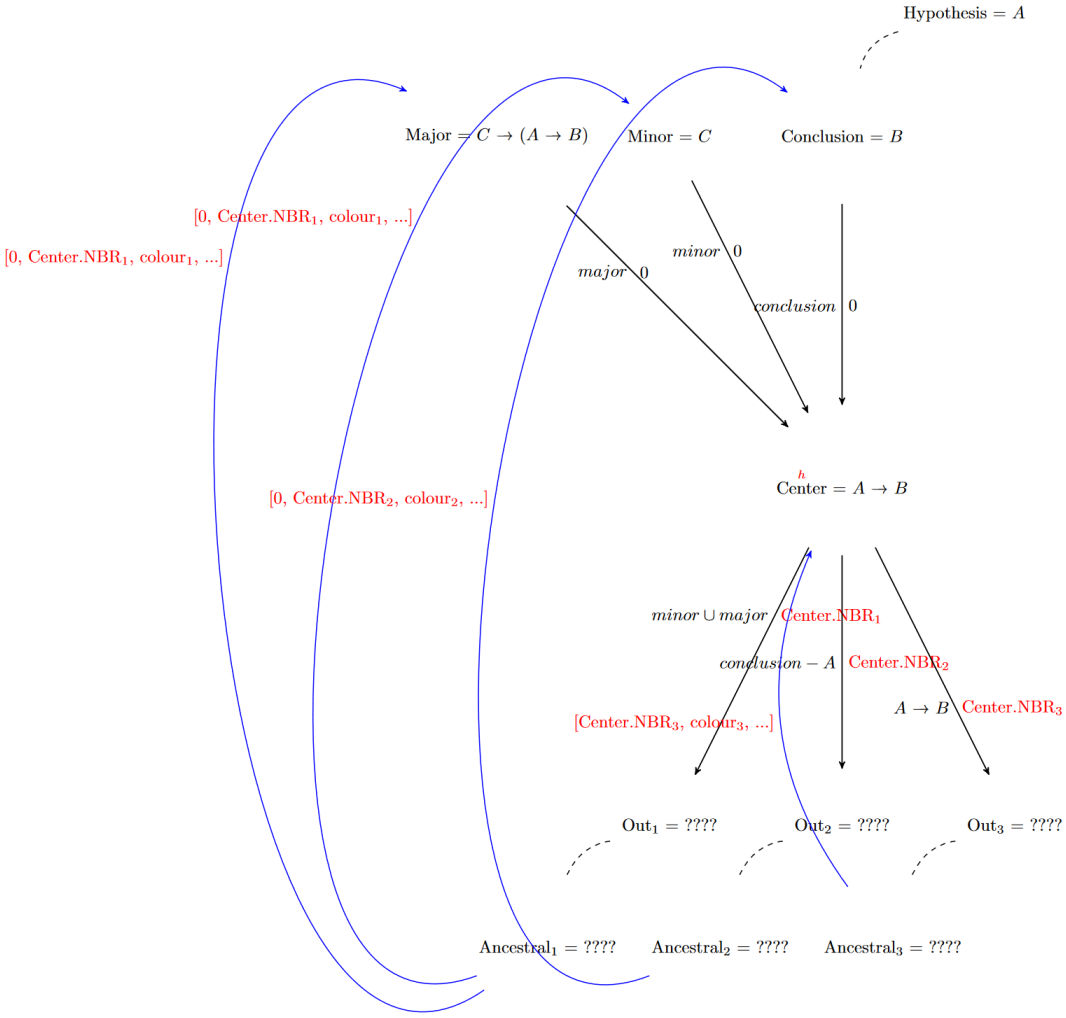

Figure 1.4: Representation of a generic Type-3 Collapse Neighborhood.

If the subset of **MUE**-rules of **HC** is applied to a valid **DLDS**, then one can be certain that it eventually halts exiting a **DLDS** that has no level with two nodes labelled with the same formula, aside from its top-formulas, because all **MUE**-rules must respect the type hierarchy.

Both these sub-groupings of the compression rules of **HC**, as well as the concept that every compression rule can be defined as to separate sub-procedures, namely the pre-collapse of each node followed by the collapse of the pair of nodes, were improvements made after the begining of the formalisation effort. The type hierarchy and division into sub-procedures were, in fact, changes that had to be made to **HC** in order to streamline the formalisation. The grouping of the rules into **MUE**-rules and **MDE**-rules was done to prove that the final **DLDS** exited after the complete run of **HC** was still simple, considering a natural separation between the ancestral and deduction edges. When talking about these improvements, it seems wise to point out that each of these groupings, these conceptualizations required subsequent changes to **HC**,

so that the algorithm would remain coherent. These changes mainly involved the addition of new rules and the reordering of both the applications of the rules and the step-by-step of the algorithm.

Though necessary for proving that the compressed proof is polynomial in size (in relation to the height and the set of all $M_{\supset}$ formulas occurring in the original tree-like derivation) proving the soundness and the coverage, as those are deeply connected in the formalisation, of the **MDE**-rules would require an entirely different set of assumptions when compared to the **MUE**-rules. Thus, the proof of these properties for the **MDE**-rules was deemed to be enough for a separate endeavor to the work of this thesis.

The Lean (lean4 v0.0.140) interactive theorem prover [4][5] was chosen for this formalization. This choice in using Lean was one made mostly out of personal familiarity with the prover and dependent type theory as a whole, the recent and continuing development of the prover itself (with lean4 having been made available relatively recently) and its active online community. Other positives, especially in comparison to some other provers, like the fact that Lean is fairly easy to run as a program (with no extraction steps required) and its more constructivist approach to theorem proving were also relevant for its choosing.

2
Previous Work

This thesis continues and furthers the work shown in [2], with its main objective being the formalisation of a Lean-assisted proof of Theorem 12 in [2]. Steps towards this result were presented at [6] and at [7], mainly concerning the proof of coverage, which was, at the time, conceptualized as a fully separate from the proof of soundness. At the time of [6] there was no distinction between **MUE**-rules and **MDE**-rules. At the time of [7] the type hierarchy was not yet fully realized and with the inductive part of the proof, the step-up from one level to the level above it, still unfinished.

This thesis also reassessed the work done in [8]. In [2] and since the authors focus on a different approach to the one used in [8], going beyond the effects of proof compression in exponentially big proofs in $M_{\supset}$ by focusing on the properties of the algorithm **HC**. The formalisation shown in [8] was of a more restrictive result than the one in [2] and thus research into its subject was discontinued.

In a series of articles, [9], [10] and [11], the authors show how to represent a derivation of α from Γ, in the natural deduction for purely minimal implicational logic ($M_{\supset}$), as a directed acyclic graph (dag). The authors also present an earlier method, in relation to the current one in [2], of converting a tree-like representation of a proof into their proposed dag-like representation[9].

Since its initial publication, this conversion procedure has evolved from a stage where the size of the dag-like derivation is polynomially bounded concerning $|\alpha| + |\Gamma|$ with a linearly bounded certificate $C(\alpha, \Gamma)$, to a stage where a certificate $C(\alpha, \Gamma)$ is not needed and that a verification that the dag-like derivation is valid can be performed in polynomial time concerning $|\alpha| + |\Gamma|$[10]. These improvements entail that $NP = PSPACE$, though this is not the focus of the work presented here. The authors would later apply their conversion procedure, now more closely resembling an implementation of **HC**, directly to a specific class of height and formula occurrences, namely the class of proofs of non-hamiltonicity for non-Hamiltonian graphs[11]. Thus, they show a proof of $CoNP = NP$ that does not need a Hudelmaier linear bound[12], though this is not the focus of the results presented here.

Since their publication, many readers of [9], [10] and [11] demanded a

computer-assisted proof of the results, due to the underlying combinatorial structure of the work being hard to follow. As such, the main objective of this thesis is to formalise such a proof, showing that the algorithm **HC** halts for every $M_{\supset}$ derivation, exiting a valid **DLDS** with no two equal nodes on the same level, aside from the top-formulas. The motivation for this thesis is the understanding that a computer-assisted proof would indeed be more verifiable, when compared to a more classic, pen-and-paper proof.

Experiments with the compression of both naive and huge proofs in natural deduction for the non-hamiltonianicity of some graphs, such as a Petersen graph, was done in [13]. A compression ratio of almost 90% was obtained, with the bigger and more redundant proofs having the best compression ratio after removing their redundant parts.

3
Proof Theory Background

Following the usual terminology of Natural Deduction and Proof Theory, this chapter describes the concepts used in the later parts of this work. Consider the usual definition of the syntax or parsing tree for formulas in purely implicational minimal logic ($\mathbf{M}_{\supset}$). Given a formula $\phi_1 \supset \phi_2$ in $\mathbf{M}_{\supset}$, ϕ_2 is its right-child and ϕ_1 its left-child. These formulas label the respective right and left child vertexes that are associated with them. A right-ancestral of a vertex v in a syntax-tree T_α of a formula α is any vertex u, such that, either v is the right-child of u, or, there is a vertex w, such that v is the right-child of w and u is right-ancestral of w. In this context, a mathematical proof in natural deduction is a derivation without open assumptions, and each of its hypotheses must be discharged by applying a specific rule[3]. The only rules applicable are the $\supset$-Introduction (or $\supset$-Intro) and $\supset$-Elimination (or $\supset$-Elim) rules. These natural deduction rules of $M_{\supset}$ are shown below:

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \supset B}\ \supset\text{-Intro} \qquad\qquad \frac{A \qquad A \supset B}{B}\ \supset\text{-Elim}$$

The left premise of an $\supset$-Elimination rule is its minor premise, and the right premise is its major premise. Both this rule's conclusion and its minor premise are sub-formulas of its major premise. Observe that the premise of an $\supset$-Introduction is also the sub-formula of its conclusion. A derivation is a tree-like structure built using $\supset$-Introduction and $\supset$-Elimination rules. The conclusion of the derivation is the root of this tree-like structure, and the leaves are called top-formulas. A proof is a derivation that has each of its top-formula discharged by an $\supset$-Introduction application. The top-formulas are also called assumptions. An assumption that it is not discharged by any $\supset$-Introduction rule in a derivation is called an open assumption. If Π is a derivation with conclusion α and $\delta_1, \ldots, \delta_n$ is the set of all of its open assumptions then it can be said that Π is a derivation of α from $\delta_1, \ldots, \delta_n$.

Definition 3.1 (Proof-Branch)**.** *A proof-branch in a proof (or derivation)* Π *is any sequence* $\beta_1, \ldots, \beta_k$ *of formula occurrences in* Π*, such that:*

– δ_1 *is a top-formula, and;*

– *For every* $i = 1, k-1$*, either* β_i *is an* $\supset$*-Elimination major premise of an application having* β_{i+1} *as conclusion or* β_i *is an* $\supset$*-Introduction premise of an application having* β_{i+1} *as conclusion, and;*

– δ_k *either is the conclusion of the derivation or the minor premise of an* $\supset$*-Elimination.*

A normal derivation in $\mathbf{M}_{\supset}$is any derivation that does not have any formula occurrence that is simultaneously the major premise of an $\supset$-Elimination and the conclusion of an $\supset$-Introduction. A formula occurrence that is, simultaneously, the conclusion of an $\supset$-Introduction and the major premise of an $\supset$-Elimination is called a maximal formula. In [3], Prawitz describes the following theorem for the Natural Deduction for the full[1] propositional fragment of minimal logic. Prawitz also describes the steps on how to obtain, via a process of reduction of the original proof/derivation, the normal proof/derivation.

Theorem 3.1 (Normalization)**.** *Let* Π *be a derivation of* α *from* $\Delta = \{\delta_1, \ldots, \delta_n\}$*. There is a normal proof* Π' *of* α *from* $\Delta' \subseteq \Delta$*. This proof* Π' *of* α *is obtained by reducing the original proof* Π *of* α*.*

In any normal proof/derivation, the format of a proof-branch provides worth information on why huge proofs are redundant, as seen in this work. Since no formula occurrence in a proof-branch can be, simultaneously, the major premise of an $\supset$-Elimination and the conclusion of an $\supset$-Introduction, the conclusion of an $\supset$-Introduction can only be either: the premise of another $\supset$-Introduction; or the minor premise of an $\supset$-Elimination; or not be the premise of any rule application at all in the proof-branch. In this last case, it must be the conclusion of the proof/derivation. Either way, it always is the last formula in the proof-branch. For any proof-branch but the principal proof-branch[2], any conclusion of an $\supset$-Introduction has to be the premise of an $\supset$-Introduction. Hence, any proof-branch in a normal proof/derivation is divided into two parts (possibly empty). For the E-part that starts the proof-branch, its top-formula and every other formula occurrence in it must be the major premise of an $\supset$-Elimination. A formula occurrence that, simultaneously, is the conclusion of an $\supset$-Elimination and the premise of an $\supset$-Introduction is called the minimal formula of the proof-branch. The minimal formula starts the I-part of the proof-branch, where every formula is the premise of an $\supset$-Introduction, except for the last formula occurrence of the proof-branch. From the format of the proof-branches, one can conclude that the sub-formula principle holds

[1]The entire propositional fragment is $\{\vee, \wedge, \supset, \neg, \bot\}$

[2]The principal proof-branch ends in the conclusion of the derivation

for normal proofs in Natural Deduction for $\mathbf{M}_{\supset}$, in fact, for many extensions of it. A proof-branch in Π is a principal proof-branch if its last formula is the conclusion of Π. A secondary proof-branch is a proof-branch that is not principal. The primary proof-branch is also called a 0-proof-branch. Any proof-branch for which its last formula is the minor premise of a rule in the E-part of a n-proof-branch is a $n+1$-proof-branch. The following corollary ensures that, without loss of generality, any Natural Deduction proof of a $\mathbf{M}_{\supset}$ tautology has only sub-formulas of its conclusion occurring in its proof-tree.

Corollary 3.1 (Sub-formula Principle)**.** *Let* Π *be a normal derivation of* α *from* $\Delta = \{\delta_1, \ldots, \delta_m\}$*. It is the case that for every formula occurrence* β *in* Π*,* β *is a sub-formula of either* α *or of some of* δ_i*.*

Without loss of generality, one can consider that some formula in $\mathbf{M}_{\supset}$ is a tautology if and only if there is a normal proof in atomic expanded form that proves it. Of course, if it is a tautology, then it must have a proof. If it has some proof, it also has a normal proof, obtained through normalization. This fact is used to obtain the expanded form of any normal proof.

Definition 3.2. *A normal proof/derivation is in expanded form, if and only if all of its minimal formulas are atomic.*

Proposition 3.1. *Let* Π *be a proof (or derivation), in* $\mathbf{M}_{\supset}$*, of* α *from* $\Gamma = \{\gamma_1, \ldots, \gamma_k\}$*. There is a proof in expanded (atomic) form of* α *from* Γ*.*

Proof. Proof of proposition 3.1. If $\varphi_1 \supset \varphi_2$ is a minimal formula in some proof-branch of Π, then one has the following form for Π, with focus on this proof-branch:

$$\begin{array}{c} \Pi_1 \\ \varphi_1 \supset \varphi_2 \\ \Pi_2 \end{array}$$

In order to have a smaller minimal formula in this proof-branch, one can replace $\varphi_1 \supset \varphi_2$ by the following derivation:

$$\dfrac{\dfrac{[\varphi_1] \qquad \varphi_1 \supset \varphi_2}{\varphi_2}}{\varphi_1 \supset \varphi_2}$$

Yielding the following new derivation:

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1 \\ [\varphi_1] \qquad \varphi_1 \supset \varphi_2\end{array}}{\varphi_2}}{\begin{array}{c}\varphi_1 \supset \varphi_2 \\ \Pi_2\end{array}}$$

Proceed by replacing, now on φ_2, until it is atomic. One can then prove by induction over the size of the minimal formula that the basis is trivial, and the figure above is the inductive one. To have the same result for the entire proof, instead of only one proof-branch, one should just do another induction over the number of proof-branches. □

The following lemma 3.1 shows that their respective top-formula uniquely defines the E-parts of any proof-branch in a normal proof in expanded form. It uses the fact that if $\gamma_1 \supset \gamma_2$ is the major premise of an application of $\supset$-Elimination, its conclusion must be to the right-hand side of this premise.

Lemma 3.1. *Let* Π *be a normal proof in expanded form. Its respective top-formulas uniquely determines the E-part of each proof-branch in* Π.

Proof. Each proof-branch in Π has the form $\{\delta_0, \ldots, q, \ldots, \delta_k\}$, where δ_0 is a top-formula, and $q = \delta_j$, an atomic formula, is the minimal formula of the proof-branch. By the proof-branch definition (definition 3.1) and the fact that Π is a normal proof, for every $i = 1, \ldots, j$, δ_{i-1} is major premise of an $\supset$-Elimination of a rule application having δ_i as conclusion. After $q = \delta_j$, the only possible rule applications are $\supset$-Introduction, by the format of normal derivation's proof-branches. So the sequence $\delta_0, \ldots, \delta_j$ is maximal. Thus, from the top-formula δ_0, one can obtain the whole sequence of formulas by picking up, recursively, the right-hand side of each of them. Finally, given the top-formula δ_0, the whole sequence $\delta_0, \ldots, \delta_j = q$ is determined. This can be proven by induction on the degree of the top-formula. □

In the lemma 3.1 above, the E-part of each proof-branch is uniquely induced from its top-formula (leaf). On the other direction, let q be an atomic formula that is a minimal formula occurring in a proof-branch $\overrightarrow{b}$ of a normal proof in expanded form. As such, q does not determine the top-formula of $\vec{b}$ uniquely. Consider the following two proof-branches, occurring in the normal and expanded proof Π, in figure 3.1, for example:

$$\{A \supset (B \supset (C \supset q)), (B \supset (C \supset q)), C \supset q, q, \ldots, \delta\}$$

$$\{B \supset (D \supset q), D \supset q, q, \ldots, \gamma\}$$

This figure shows a relation, pointed by the dashed lines, between the minimal formula occurrences in Π and their respective occurrences in the associated syntax tree. The minimal formula q occurs in both proof-branches, and if given only q, it is impossible to uniquely determine the top-formula of the proof-branch to which q belongs. Nevertheless, upon more careful consideration,

these q's in the proof-branches are not the same. In the first example, q is a sub-formula of $A \supset (B \supset (C \supset q))$, while in the second proof-branch, q is a sub-formula of $B \supset (D \supset q)$. If Π is a normal proof of α, then these q's point to different occurrences in the syntax tree of α. On the other hand, given a syntax tree T_α of α and an atomic formula q, it is known that q is a leaf in T_α. There can be more than one leaf labeled with q, but given a specific leaf q, one can always determine from which top-formula it can be derived. The following lemma states that this must be true concerning any normal proof Π. First observe the following fact, stated here as a lemma 3.2 without any proof. Then, as a consequence of said lemma, consider the following corollary 3.2.

Lemma 3.2. *Any formula in* $\mathbf{M}_\supset$ *is of the form* $(\alpha_0 \supset (\alpha_1 \supset \ldots (\alpha_k \supset q)\ldots)$, *where* q *is atomic.*

Corollary 3.2. *If* Π *is a normal proof in expanded form and* q *is the minimal formula of a proof-branch* $\overrightarrow{b}$ *then the top-formula of this proof-branch is of the form* $(\alpha_0 \supset (\alpha_1 \supset \ldots (\alpha_k \supset q)\ldots)$, *for some* α_i, $i = 1, k$.

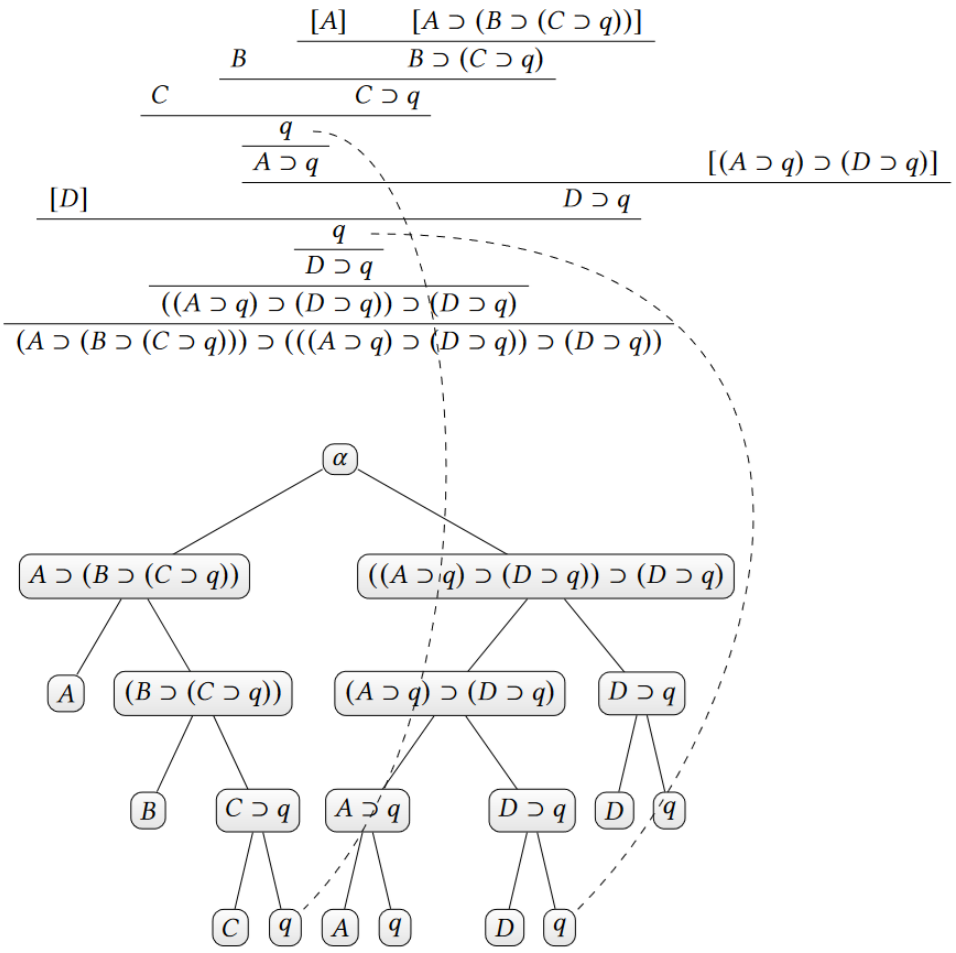

Figure 3.1: Relating minimal formulas and leaves in the syntax tree of a tautology

In any $M_{\supset}$ natural deduction derivation, any application of an $\supset$-Introduction rule has a way to indicate which formula occurrences are discharged by the application of this $\supset$-Introduction application. One way to formalize this indication is to add edges (discharging edges) linking the conclusion of the rule applies to each discharged formula occurrence in the derivation tree that represents the natural deduction derivation[3]. This may be a convenient representation in many formulations of natural deduction. However, to not crowd the dag-like derivations with unnecessary information, these discharging edges are dropped out by assigning to each deduction-edge the string of bits that represents the set of assumptions from which the formula that labels the target of this deduction-edge depends on. This is formalized below.

Definition 3.3. *Let α be an implicational formula, $Sub(\alpha)$ the set of subformulas of α, and $\mathcal{O}(\alpha) = \{\beta_0, \beta_1, \ldots, \beta_k\}$ a linear ordering on $Sub(\alpha)$. A bit-string on $\mathcal{O}(\alpha)$ is any string $b_0 b_1 \ldots b_k$, such that $b_i \in \{0, 1\}$, for each $i = 0, 1, \ldots, k$.*

There is a bijective correspondence between bit-strings on $\mathcal{O}(\alpha)$ and sets of subformulas of α, given by $Set(b_0 b_1 \ldots b_k) = \{\beta_i / b_i = 1\}$. The bit-string on $\mathcal{O}(\alpha)$ will be used to drop out the discharging edges and make explicit the information on formula dependencies in a derivation. The set of bit-strings on $\mathcal{O}(\alpha)$ is denoted by $Bits(\alpha, \mathcal{O}(\alpha))$. The inverse function of Set is well-defined, for a fix ordering on $Sub(\alpha)$ and is denoted by Set^{-1}. The set of all bit-strings on a set S, under ordering $\mathcal{O}_S$ is denoted by $\mathcal{B}(\mathcal{O}_S)$.

The following definition is used to argue that even when considering a restricted form of $\supset$-Intro rules (greedy), the set of theorems is not changed. This restricted form of an $\supset$-Intro is used to provide a sound way to remove all the discharging edges from the tree-like proofs.

Definition 3.4. *Consider a derivation Π of β having Δ as assumptions. Let $\alpha \in \Delta$ be an (open) formula assumption in Π. Applying an $\supset$-Introduction in Π is greedy, iff it produces $\alpha \supset \beta$ as conclusion and discharges in Π every open occurrence of α from which its premise β depends on.*

Applying an $\supset$-Introduction to a tree-like derivation is greedy, if and only if, its corresponding application in a natural deduction derivation is also greedy [3]. Reaffirming the terminology used in [3], a proof is a derivation that has no open assumption, i.e., all hypotheses are discharged by some $\supset$-Intro, and that a branch is a complete sub-derivation (sub-tree). In a proof,

[3]It can be noted that assigning unique marks (numbers, for example) to each formula occurrence in a derivation and attach to each $\supset$-Introduction application the set of marks associated to the set of its discharged formula occurrences is equivalent to add edges indicating these discharges.

every hypothesis is discharged by some $\supset$-intro rule application, and hence any branch is a complete sub-derivation (sub-tree). The algorithm below modifies a given proof in $M_{\supset}$ into a greedy proof in $M_{\supset}$:

Algorithm 1 Greedy Proof Conversion

Require: A proof Π in $M_{\supset}$, where n is the level of the highest branch in Π

1: $j = n$
2: **while** $0 < j$ **do**
3: **for** each branch B of level j in Π **do**
4: replace each intro-app downwards by a greedy intro-app
5: possibly discharging more formula occurrences in B
6: **end for**
7: $j = j - 1$
8: **end while**

In [2] the authors also demonstrate the following lemma about the above procedure and preservation of the completeness of greedy proofs:

Lemma 3.3. *Greedy $\supset$-Intro is Complete: Let Π be a proof of a $M_{\supset}$ formula α. If Π' is the result of the above procedure applied to Π, then Π' is also a valid proof of α in $M_{\supset}$.*

In [2] the authors also prove that any greedy proof of α is mapped to a rooted, leveled, and labeled dag-like proof of α, where its root is labeled with α. The tree-like dependency is also mapped into this initial dag-like proof. The article also argues that **HC**, when applied to any dag-like proof, preserves the logical information provided by the decorations used in the dag, resulting in the preservation of the soundness of the dag-like proof. However, this demonstration is lengthy and must go through multiple cases.

4
Horizontal Compression

4.1
Primary Definitions

In [2] (sections 3 and 5), in a series of hand-proven results, the authors show that any natural deduction greedy proof of an $M_{\supset}$ formula α can be mapped to a **DLDS** having its root labelled with α. This **DLDS** represents the underlying tree of the natural deduction greedy proof, but instanced with all the decorations that a **DLDS** needs. The tree-like dependency of the natural deduction greedy proof is also mapped into this **DLDS** proof.

In [2], the authors also argues that **HC**, when applied to any **DLDS** proof, preserves the logical information provided by the decorations used in the **DLDS**, resulting in the preservation of soundness of the proof. For the purposes of this thesis, **HC** takes as input an arbitrary **DLDS**, defined as follows:

Definition 4.1 (Dag-Like Derivability Structures)**.** *Let* Γ *be a set of* $M_{\supset}$ *formulas,* $\mathcal{O}_\Gamma$ *an arbitrary linear ordering on* Γ*, such that* $n > 0$*, for* $n \in \mathcal{O}_\Gamma$*, and* $\mathcal{O}_\Gamma^0 = \mathcal{O}_\Gamma \cup \{0, \lambda\}$*. A* **D***ag-***L***ike* **D***erivability* **S***tructure, is a tuple* $\langle V, (E_D^i)_{i \in \mathcal{O}_\Gamma^0}, E_A, r, l, L, P\rangle$*, where:*

- V is a non-empty set of nodes;
- For each $i \in \mathcal{O}_\Gamma^0$, $E_D^i \subseteq V \times V$ is the family of sets of edges of deduction;
- $E_A \subseteq V \times V$ is the set of edges of ancestrality;
- $r \in V$ is the root of the **DLDS**;
- $l : V \to \Gamma$ is a function, such that, for $v \in V$, $l(v)$ is the (formula) label of v;
- $L : \bigcup_{i \in \mathcal{O}_\Gamma^0} E_D^i \to \mathcal{B}(\mathcal{O}_S)$ is a function, such that, for every $\langle u, v\rangle \in E_D^i$, $L(\langle u, v\rangle)$ is the bitstring representing from which formulas the i-th colored deduction-edge $\langle u, v\rangle$ carries its dependency;
- $P : E_A \to \{1, \ldots, || \Gamma ||\}^\star$, such that, for every $e \in E_A$, $P(e)$ is a string of the form $o_1; \ldots; o_n$, where each o_i, $i = 1, \ldots, n$, is an ordinal in $\mathcal{O}_\Gamma$.

For each $i \in \mathcal{O}_\Gamma^0$ and $\langle u, v\rangle \in E_D^i$, i is called the color of the edge $\langle u, v\rangle$. Each deduction-edge is coloured with formulas from Γ or the colour 0. The colours are introduced every time a collapsing of nodes, as explained in [2], is performed. Tree-like greedy derivations have only 0 coloured deduction edges. **DLDS**s obtained from Tree-like greedy derivations by effectively collapsing vertexes, sometimes edges, have coloured deduction edges. Not all **DLDS** is in the image of the function that maps Tree-like derivations into **DLDS**, but any natural deduction (usually tree-like) derivation of a formula α can be seen as a **DLDS**, as shown in [2]. It is also worthy of note that every **DLDS** having E_A empty and only E_D^i for $i = 0$, i.e., a **DLDS** without ancestral-paths (or ancestral-edges) and only with 0-coloured deductive edges, is a greedy tree-like derivation[1].

4.2 The Horizontal Compression Algorithm and Rules

In [2], the authors provide an algorithm, named **HC**, for obtaining a compressed dag-like proof, meaning a proof represented by a **D**irected **A**cyclic **G**raph, of any purely implicational minimal tautology. This dag-like proof has more decoration elements and labels than regular proofs in purely implicational minimal logic ($M_\supset$) and, using these elements, a verification that the dag-like proof is valid can be done in polynomial time [2]. The authors named this type of dag-like proof a **D**ag-**L**ike **D**erivability **S**tructure (**DLDS**), which is defined in [2] and also in accordance to the other definitions mentioned in this text.

The horizontal compression mentioned in the name of the algorithm is composed of a series of horizontal collapses. A horizontal collapse applies to a dag-like decorated greedy derivation. It aims to identify two or more nodes in the rooted dag-like derivation at the same deduction level. The collapsing applies from the conclusion level, namely the zero level, towards the assumptions levels. When applied to tree-like rooted and decorated derivations, it yields dags instead of trees. The following algorithm formally defines this operation as a case analysis, comprised of a set of collapse rules (or compression rules) and that applies to a dag-like derivation to yield a (new) dag-like derivation.

[1] Greedy tree-like derivations are defined in [2].

Algorithm 2 Horizontal Compression

Require: A tree-like greedy derivation $\mathcal{D}$

Ensure: That the $DLDS$ is $\mathcal{D}$ compressed

```
for l from 1 to h(D) do
    for u and v at l do
        HCom(u, v)
    end for
end for
```

The horizontal collapsing initially transforms tree-like derivations into dag-like derivations. Additional structure is needed to allow us to verify that a particular dag-like derivation is a (correct) derivation, indeed. A dag-like derivability structure is the underlying structure used to encode dag-like derivations. Thus, a dag-like derivation is a $DLDS$ instance, as defined in the above definition, and a condition that should be true about this $DLDS$ instance.

Figure 4.1 shows the first rule, named Rule **R0IE**. The figure shows how to read the pictorial representation of each horizontal compression rule. Both the left and the right-hand sides are subgraphs. In the left-hand side of the rule in Figure 4.1, p_i, $i = 1, 3$, u, v, and the two bullets ($\bullet$) below them are all pairwise different nodes in the subgraph, such that $l(v) = l(u)$.[2] The deductive edges are in black and have as labels the bit-string representing the dependency set denoted by L. For example, $L(\langle p_1, u\rangle) = \bar{c}_1$ shows that the deductive edge $\langle p_1, u\rangle \in E_d^0$ is labeled by the dependency set $Sets(\bar{c}_1)$. The absence of a label on an edge indicates that the edge is unlabeled. A label's node is $\bullet$ whenever it is not relevant what is its label to read the rule. Edges that belong to E_D^i have the colour i; this is the red ordinal number $1, \ldots, n$ on a black deduction edge. The members of E_A, the ancestor edges, are coloured blue, and their labels under P labelling function are red in the picture. For example, $\langle \bullet, p_1\rangle \in E_A$ and $P(\langle \bullet, p_1\rangle) = 1$. Moreover, $\langle u, \bullet\rangle \in E_D^1$ in the graph can be found to the right-hand side of **Rule 1**. Dashed black lines represent paths in the graph composed solely of deduction edges. For the sake of clarity, Rule **R0IE** is classified as a **MUE**-rule, for **M**oving **U**pwards **E**dges, because it produces ancestral-edges arriving above the collapsed nodes.

[2]The reader should know that in all graphical representations of the rules, both in this paper and in [2], nodes and edges drawn in different positions are always assumed to be different from one another.

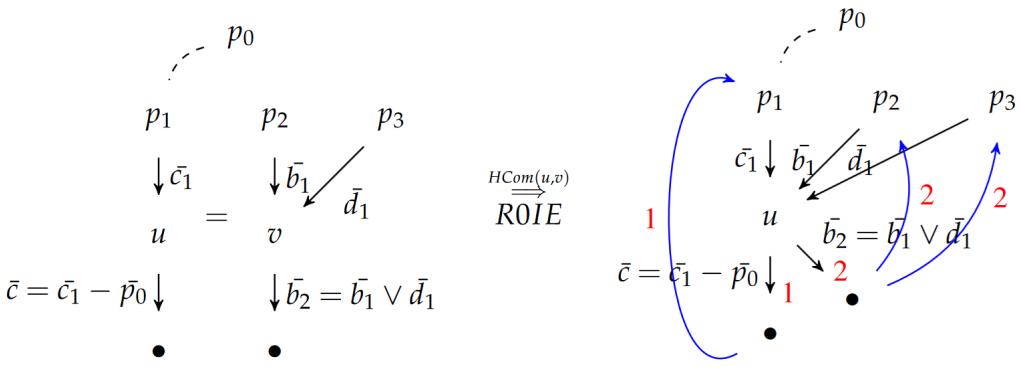

Figure 4.1: Nodes u and v before (left) and after (right) collapse HCom(u, v)

4.3 The Preservation of Soundness of the Compression Rules

Some further definitions concerning the compression rules and how they affect a *DLDS* must be made. All of these definitions were taken from [2], and are written here in order to have a more self-contained document.

Definition 4.2. *Incoming Deductive Edges of a node: Given a DLDS $\mathcal{D}$ of α from Γ and a node $k \in \mathcal{D}$, the deductive in-degree of k is defined as* $INS(k) = \{f : f \in E_D^i, i \in \mathcal{O}(\Gamma \cup \{\alpha\})^0 \wedge target(f) = k\}$.

Definition 4.3. *Outgoing Deductive Edges from a node: Given a DLDS $\mathcal{D}$ of α from Γ and a node $k \in \mathcal{D}$, the deductive out-degree of k is defined as* $OUTS(k) = \{f : f \in E_D^i, i \in \mathcal{O}(\Gamma \cup \{\alpha\})^0 \wedge source(f) = k\}$.

Note that for any node k, both sets, $INS(k)$ and $OUTS(k)$, do not take into account the ancestor edges in their definition. Note that the members of E_A are not deductive edges. However, they play an important, though auxiliary role in the logical reading of any *DLDS*. A simple observation is that there is a natural map from a **D**ecorated **G**reed **T**ree-Like **D**erivation (*DGTD*) to a *DLDS*.

Definition 4.4. *Definition 6: Let $\mathcal{T} = \langle V, E_D, E_d, r, l, L\rangle$ be a DGTD. Let $\mathcal{O}_S$ be the order on the range of l, provided by $\mathcal{T}$ itself and Γ the set of leaves in $\mathcal{T}$. Let $Dag(\mathcal{T})$ be $\langle V, (E_D^i)_{i\in\mathcal{O}_\Gamma^i}, E_A, r, l, L, P\rangle$, where $E_D^0 = E_D$, $E_D^i = \emptyset$, for all $i \neq 0$ and $E_A = \emptyset$, $P = \emptyset$.*

It is easy to verify that $Dag(\mathcal{T})$ is well-defined, and hence it is a *DLDS*, for every *DGTD* $\mathcal{T}$. Thus, there is the mapping *Dag* from a *DGTD* to a *DLDS*. When reading a *DGTD* from top to bottom in a tree-like Natural Deduction derivation, there is at most one path from any top-formula occurrence to any other formula occurrence in the derivation. The following fact is an easy consequence of *Dag*'s definition above.

Proposition 4.1. *Let $\mathcal{T}$ be a DGTD. For every pair of nodes v and u in $\mathcal{T}$, there is a bijection between the paths from v to u, in $\mathcal{T}$, and 0-paths, i.e., using only members of E_D^0, from v to u in $Dag(\mathcal{T})$. Moreover, the dependency sets, assigned by L, in both structures, are equal for every edge $\langle v, u\rangle \in E_D^0$.*

From the definition of the mapping Dag, one can see that there is no path in the $DLDS$ $Dag(\mathcal{T})$ with colors different from 0, due to $E_A^i = \emptyset$, for all $i \neq 0$. Moreover, there are no paths in E_A, for $E_A = \emptyset$.
The following definition shows how the information stored in the component P, the seventh one, the last, of any $DLDS$ is used as a relative address for nodes in it. It uses:

Definition 4.5. *$el(\{e\}) = e$ and $el(S) = \perp$, if S is not $\{a\}$ for some a.*

Definition 4.6. *Relative Address of a Node: Let $\mathcal{D}$ be a $DLDS$ of α from Γ and let $\gamma \in \mathcal{O}(\Gamma \cup \{alpha\})^0)^\star$. Then γ is the address of a node $v \in \mathcal{D}$ relative to a node $u \in \mathcal{D}$ iff the following algorithm 3 returns v on input γ, $\mathcal{D}$ and u. The underlying idea is that γ provides information on every branching in the path from u downwards v. Each ordinal from left to right in γ indicates which branch to take in.*

Algorithm 3 Finding a Node from its Relative Address and Origin of the Path

Require: u, the origin, $\mathcal{D}$, the $DLDS$, and the relative address γ

1: $b \leftarrow u$
2: $glues \leftarrow \gamma$
3: **while** $glues \neq \epsilon$ **do**
4: **if** $size(OUTS(b)) == 1$ **then**
5: $g \leftarrow el(OUTS(b))$
6: $b \leftarrow target(g)$
7: **else if** $size(OUTS(b)) > 1 \wedge size(\{e/(e \in OUTS(b)) \wedge (color(e) = head(\gamma))\}) = 1$ **then**
8: $g \leftarrow el(\{e/(e \in OUTS(b)) \wedge (color(e) = head(\gamma))\})$
9: $b \leftarrow target(g)$
10: $glues \leftarrow rest(\gamma)$
11: **else**
12: Return false
13: **end if**
14: **end while**
15: Return b

The definition of Deductive Path, found below, is used to discern when a $DLDS$ corresponds to a valid derivation.

Definition 4.7. *Deductive Path: Given two nodes v_1 and v_2 in a* **DLDS** $\mathcal{D} = \langle V, (E_D^i)_{i\in\{\bar{\lambda}\}\cup\mathcal{O}_\Gamma^i}, E_A, r, l, L, P\rangle$*, a path $e_1, e_2, \ldots, e_n$ from v_1 to v_2 is called a deductive path, iff, for each $p = 1, \ldots, n$, $e_p \in \bigcup_{i\in\{\bar{\lambda}\}\cup\mathcal{O}_\Gamma^i} E_D^i$. In particular, if $e_1, e_2, \ldots, e_n$ is a deductive path from v_1 to v_2 and there is $i \neq 0$, such that $e_j \in E_D^i$ or $e_j \in E_D^{\bar{\lambda}}$, for some $0 \leq j \leq n$, then the path is a mixed deductive path from v_1 to v_2.*

Given a $DLDS$ $\mathcal{D} = \langle V, (E_D^i)_{i\in\mathcal{O}_\Gamma^i}, E_A, r, l, L, P\rangle$ and a node $w \in V$, define: **Pre(w)** $= \{v$: Such that there is a deductive path from v to $w\}$, as the set of nodes that are linked to w by some deductive path; **Top(w)** $= \{v$: Such that $v \in Pre(w)$ and either v is marked as hypothesis, or there is no $v' \in V$, or $\langle v', v\rangle \in (E_D^i)_{i\in\mathcal{O}_\Gamma^i}\}$, as the set of top nodes of a $DLDS$; **DedPaths(w)** $= \{\langle e_1, \ldots, e_n\rangle$: Such that $e_1 \ldots e_n$ is a deductive path, with $source(e_1) \in TopNode(w)$ and $target(e_n) = w\}$, as the set of full deductive paths reaching to $w \in V$.

Definition 4.8. *Relation $\sim$ between Dependency Sets: For any pair of dependency sets $\bar{b}$ and $\bar{c}$, $\bar{b} \sim \bar{c}$ holds, if and only if, $\bar{c} = \bar{b}$ or $\bar{c} = \lambda$ or $\bar{b} = \lambda$.*

Definition 4.9. *Given a DLDS* $\mathcal{D} = \langle V, (E^i_D)_{i\in\mathcal{O}^i_\Gamma}, E_A, r, l, L, P\rangle$ *and a node* $w \in V$*, define* $Flow(\mathcal{D}, w)$ *as a function from* $Pre(w)$ *into* $\wp((\mathcal{O}^0_\Gamma)^* \times \mathcal{B}(\mathcal{O}_S))$*, such that:*

$$Flow(\mathcal{D}, w)(v) =$$

$$\begin{cases}
\{(b(l(\vec{v})), P(\langle v', v\rangle)) : \langle v', v\rangle \in E_A,\ v' \in V\} & \text{if } v \in Top(w), \text{ there is } v' \in V,\ \langle v', v\rangle \in E_A \\
\{(b(l(\vec{v})), 0)\} & \text{if } v \in Top(w) \text{ and } \nexists v' \in V,\ \langle v', v\rangle \in E_A \\
\left\{(\vec{b_1} \vee \vec{b_2}, p) : \boxed{\begin{array}{l}(v_1, v_2, v) \in \supset_E \text{ and} \\ (b_i, [o_i|p]) \in Flow(\mathcal{D}, w)(v_i), \\ \text{and } \langle v_i, v\rangle \in E^{o_i}_D, \text{ and,} \\ b_i \sim L(\langle v_i, v\rangle),\ i = 1, 2, \\ \text{OR } (v_1, v_2, v) \in \supset_E \text{ and} \\ (b_i, [0|p]) \in Flow(\mathcal{D}, w)(v_i),\ i = 1 \text{ or} \\ i = 2, \text{ and} \\ (b_j, \emptyset) \in Flow(\mathcal{D}, w)(v_j),\ j \neq i, \text{ and} \\ \langle v_i, v\rangle \in E^0_D,\ \langle v_j, v\rangle \in E^0_D, \text{ and} \\ b_i \sim L(\langle v_k, v\rangle),\ k = 1, 2, \end{array}}\right\} \\
\cup \\
\left\{(\vec{b'} - \vec{\alpha}, p) : \boxed{\begin{array}{l}(v', v) \in \supset_I \text{ and} \\ (b', [o'|p]) \in Flow(\mathcal{D}, w)(v') \text{ and} \\ \vec{b'} \sim L(\langle v', v\rangle) \text{ and,} \\ \langle v', v\rangle \in E^{o'}_D, \text{ and } l(v) = \text{“}\alpha \supset l(v')\text{”}\end{array}}\right\} \\
\cup \\
\left\{(b(l(\vec{v})), 0) : v \text{ is marked with } \hbar \text{ and } \nexists v',\ \langle v', v\rangle \in E_A\right\} \\
\cup \\
\left\{(b(l(\vec{v})), P(\langle v', v\rangle)) : v \text{ is marked with } \hbar \text{ and } \langle v', v\rangle \in E_A\right\} \\
\cup \\
\left\{(\vec{b'} - \vec{\alpha}, P(\langle v_a, v\rangle)) : \boxed{\begin{array}{l}(v', v) \in \supset_I \text{ and} \\ (b', 0) \in Flow(\mathcal{D}, w)(v') \text{ and} \\ v_a \in V,\ \langle v_a, v\rangle \in E_A \text{ and,} \\ P(\langle v_a, v\rangle) = [j \mid p] \text{ and } \langle v, v''\rangle \in E^j_D \\ v'' \in Pre(w) \text{ or } v'' = w \\ \text{and } v_a \in Pre(w) \\ \vec{b'} \sim L(\langle v', v\rangle), \text{ and} \\ l(v) = \text{“}\alpha \supset l(v')\text{”}, \text{ and} \\ \langle v', v\rangle \in E^0_D \end{array}}\right\} \\
\cup \\
\left\{(\vec{b_1} \vee \vec{b_2}, j) : \boxed{\begin{array}{l}(v_1, v_2, v) \in \supset_E \text{ and} \\ (b_i, 0) \in Flow(\mathcal{D}, w)(v_i), i = 1, 2 \\ \text{and } v_a \in V,\ \langle v_a, v\rangle \in E_A, \text{ and} \\ P(\langle v_a, v\rangle) = [j \mid p] \text{ and } \langle v, v'\rangle \in E^j_D \\ v' \in Pre(w) \text{ or } v' = w \\ \text{and } v_a \in Pre(w) \\ \langle v_i, v\rangle \in E^0_D\ i = 1, 2, \text{ and} \\ b_k \sim L(\langle v_k, v\rangle),\ k = 1, 2 \end{array}}\right\}
\end{cases}$$

Definition 4.10. *Given a structure* $\mathcal{D} = \langle V, (E^i_D)_{i\in\mathcal{O}^i_\Gamma}, E_A, r, l, L, P\rangle$*, it is a valid DLDS, iff, the following conditions hold on it:*

- ➡ **Color-Acyclicity** *For each* $i \in \mathcal{O}^i_\Gamma$*,* E^i_D *does not have cycles;*

➡ **Color-Leveled** *The rooted sub-dag* $\langle V, (E_D^i)_{i\in\mathcal{O}_\Gamma^i}, r\rangle$ *is leveled;*

➡ **Ancestor Edges** *For each* $\langle v_1, v_2\rangle \in E_A$*, the level of* v_1 *is smaller than the level of* v_2*;*

➡ **Ancestor Backway Information** *For each* $\langle v_1, v_2\rangle \in E_A$*,* $P(\langle v_1, v_2\rangle)$ *is the relative address of* v_1 *from* v_2*;*

➡ **Simplicity** *The rooted sub-dag* $\langle V, (E_D^i)_{i\in\mathcal{O}_\Gamma^i}, r\rangle$ *is a simple graph, i.e, for each pair of nodes* v_1 *and* v_2*, there is at most an* $i \in \mathcal{O}_\Gamma^i$*, such that* $\langle v_1, v_2\rangle \in E_D^i$*;*

➡ **Ancestor-Simplicity** *The sub-dag* $\langle V, E_A\rangle$ *is a simple graph;*

➡ **Non-Nested Ancestor Edges** *For each* $\langle v_1, v_2\rangle \in E_A$*, there is no* w *in the path from* v_2 *to* v_1*, determined by* $P(\langle u, v\rangle \in E_A)$*, such that* $\langle w, z\rangle \in E_A$*, for some* $z \in E_A$*;*

➡ **Correct Rule Application** *For each* $w \in V$*,* $Flow(\mathcal{D}, w)(v)$ *is well-defined for each* $v \in Pre(w)$*. Moreover, for each* w *and* v*,* $Flow(\mathcal{D}, w)(v)$*, with* $v \in Pre(w)$*, one has:*

- *If* $Flow(\mathcal{D}, w)(v) = \{(\vec{b}, p)\}$ *then* $OUT(v) = \{\langle v, v'\rangle\}$ *and the color of* $\langle v, v'\rangle$ *is* $head(p)$*, i.e.,* $\langle v, v'\rangle \in E_D^{head(p)}$*, and* $\vec{b} = L(\langle v, v'\rangle)$*, and;*
- *If* $Flow(\mathcal{D}, w)(v) \neq \emptyset$ *and it is not a singleton either then for each* $\Phi_i = \{(\vec{b}, p) \in Flow(\mathcal{D}, w)(v) : head(p) = i\}$*:*
  - *If* $\Phi_i \neq \emptyset$ *then there is only one* v' $\langle v, v'\rangle \in E_D^i$ *and if* $\Phi_i = \{(\vec{b}, p)\}$ *then* $L(\langle v, v'\rangle) = \vec{b}$ *else* $L(\langle v, v'\rangle) = \lambda$*, and;*
  - *If* $\Phi_i = \emptyset$ *then there is no* $v' \in V$*, such that,* $\langle v, v'\rangle \in E_D^i$*.*

Each item in this last definition is an invariance property that should be preserved by all compression rules applications. It is worth noting that in **Correct Rule Application**, the verification that a rule application is correct involves, among other things, finding out that the premises agree with the conclusion and checking that the dependency sets are correctly assigned, this is the main role of function *Flow*.

5
Example of Horizontal Compression

A natural deduction proof of a $M_{\supset}$ tautology can end in a series of $\supset$-Introduction rule applications, and it must if one considers normal proofs. The **HC** algorithm collapses equal formulas in each level from the bottom up and left to right. The final part cannot be compressed with only $\supset$-Introduction, because it already has one formula by level. So, to show smaller natural deduction derivations, one must consider derivations without only the final $\supset$-Introduction part. For example, the derivation in figure 5.1 comes from a proof of $A_1 \supset A_2 \supset (A_1 \supset (A_2 \supset A_3)) \supset (A_2 \supset (A_3 \supset A_4)) \supset (A_3 \supset (A_4 \supset A_5)) \supset (A_1 \supset A_5)$ that that had the final $\supset$-Introduction part removed.

Bitstrings are used here to represent subsets of a linearly ordered finite set. According this, the subset $\{A \supset B, B \supset C, C\}$ of the the ordered set $\{A, B, C, A \supset B, B \supset C\}$, with order $A \prec B \prec C \prec A \supset B \prec B \supset C$, is represented by 00111. This representation is given below, in definition 5.1.

Definition 5.1. *Let L be a set of formulas in $M_{\supset}$ and $\mathcal{O}(\alpha)$ be a linear order $\mathcal{O}(L) = \{\beta_0, \beta_1, \ldots, \beta_k\}$. A bitstring on $\mathcal{O}(L)$ is any string $b_0 b_1 \ldots b_k$, such that $b_i \in \{0, 1\}$, for each $i = 1 \ldots k$. There is a bijective correspondence between bitstrings on $\mathcal{O}(\alpha)$ and subsets of L, given by $Set(b_0 b_1 \ldots b_k) = \{\beta_i / b_i = 1\}$.*

The derivation in figure 5.1 is greedy, i.e., all $\supset$-Introduction rule applications concluding $\alpha \supset \beta$ discharge every possible occurrence of α that is a hypothesis of the derivation of its premise, β. Natural Deduction Greedy Derivations can be represented as labelled trees. The tree nodes are labelled by the formulas in the derivation, and the edges are labelled by bitstrings that represent sets of formulas. In figure 5.2, the following linear ordering $\prec$ on the set of formulas that are in figure 5.1 is to be considered:

$$A_1 \prec A_2 \prec A_3 \prec A_4 \prec A_5 \prec$$

$$\prec A_1 \supset A_2 \prec A_2 \supset A_3 \prec A_4 \supset A_5 \prec A_1 \supset A_5 \prec$$

$$\prec A_1 \supset (A_2 \supset A_3) \prec A_2 \supset (A_3 \supset A_4) \prec A_3 \supset (A_4 \supset A_5)$$

One can choose any linear order to represent natural deduction derivation in the form of trees. The (fixed) linear order is used to represent sets of formulas

as bitstrings. The linear order must be taken as part of the representation of the greedy natural deduction tree representation. For example, the set $\{A_1, A_1 \supset A_2, A_1 \supset (A_2 \supset A_3)\}$ is represented by the bitstring 100001000100.

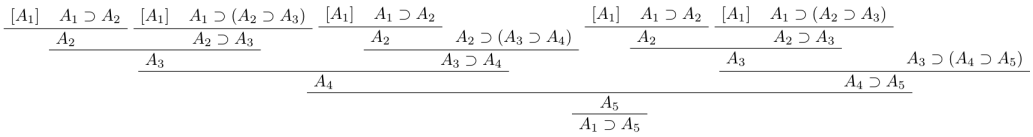

Figure 5.1: Deriving $A_1 \supset A_5$ from $A_1 \supset A_2$, $A_1 \supset (A_2 \supset A_3)$, $A_2 \supset (A_3 \supset A_4)$, and $A_3 \supset (A_4 \supset A_5)$.

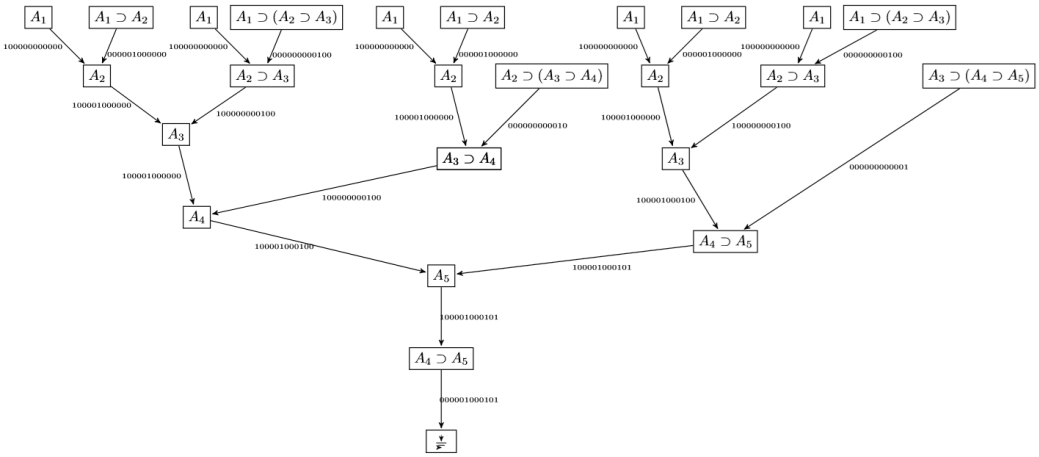

Figure 5.2: **DLDS** equivalent to the initial tree-like derivation.

The algorithm **HC** can be used to compress the derivation in figure 5.2. The algorithm obtains smaller representations of proofs and derivations in minimal implicational logic by applying transformation rules to a given greedy derivation or proof that must be provided as a labelled tree, as in figure 5.2. The rules are applied deterministically, bottom-up and left to right. The purpose of each rule is to collapse redundant parts of the derivation. They collapse nodes that have the same label in the same level. For example, the formula A_3 appears twice in level 3[1]. These two occurrences of A_3 are conclusions of identical derivations. Moreover, in level 4 three repeated occurrences of A_2 can be collapsed too. It first collapses these two lower occurrences of the formula A_3 using the rule depicted in figure 5.3, whose official name is **R0EE**. It matches the tree in level 3 according to the markings in figure 5.4. The nodes labelled with A_3 match with u and v, in the rule, respectively, and their respective children p_1 and p_4, and, p_2, p_3 match with the premisses in the rule accordingly.

[1]The root of a proof/derivation is in level zero

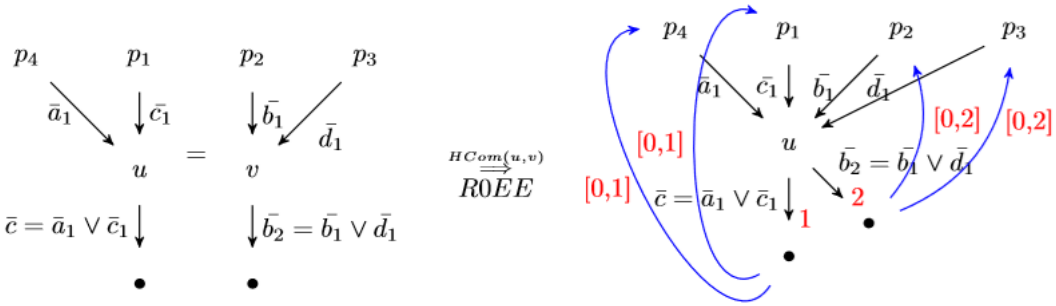

Figure 5.3: **MDE** Compression Rule **R0EE**.

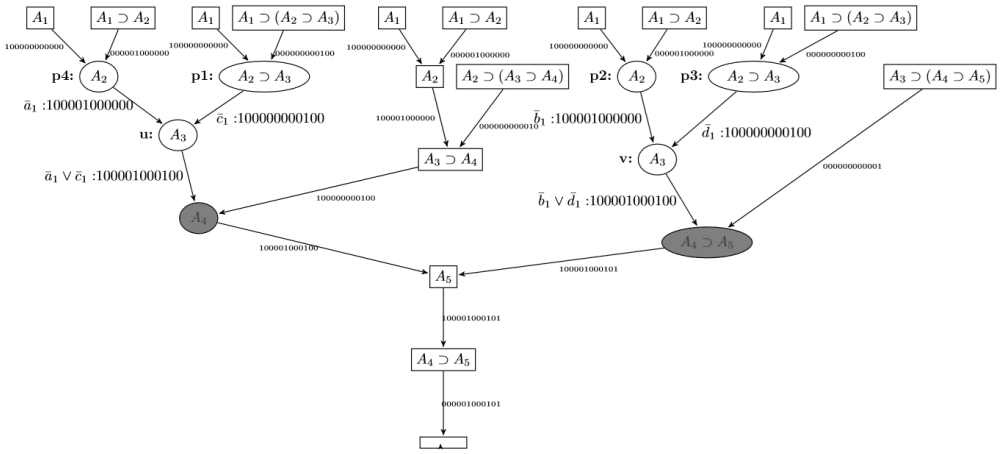

Figure 5.4: Collapsing of nodes u and v: Matching of **R0EE**.

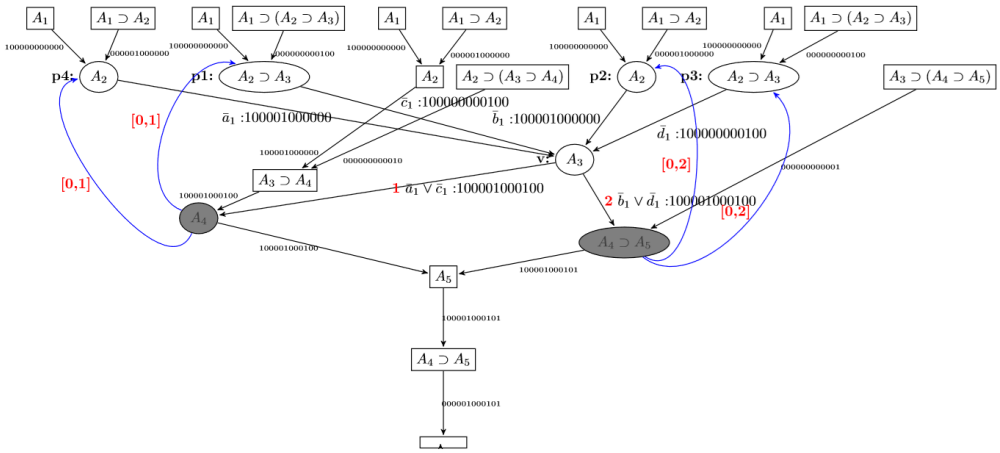

Figure 5.5: Resulting **DLDS** after collapse by **R0EE**.

All the compression rules of **HC** are deterministic and used to define a rewriting operation, $HCom(u, v)$, that collapses two nodes, u and v that are labeled with the same formula. Each rule has a context provided by pattern matching and applies to a specific graph, afterwards defined as a **DLDS**[2] $\mathcal{D}$, by the matching of its left-hand side.

[2]Dag-Like Decorated Structure

The rules are named as Rim_lm_r, where $i = 0 \ldots 3$ and $m_l, m_r \in \{I, E, H, X\}$, such that i is the type of the rule itself and m_l/m_r is type of the left/right vertex to be collapsed. For exemple, in figure 5.3, the rule named **R0EE**, collapses the conclusion of an application of the $\supset$-Elimination natural deduction rule with the conclusion of another application of the $\supset$-Elimination natural deduction rule. There are also small variations in this naming convention, with rules such as $R_v2m_lm_r$ and $R_e2m_lm_r$. The indexes v and e indicate that the respective rule collapses only vertexes (v) or vertexes and edges (e). Rules **R_v2EE**, figure 5.6, and **R_e2EE**, figure 5.8, depict these examples. In this introductory example it is not important to understand what a **DLDS** is in detail, but the fact that they represent derivations, possibly in the form of a *DAG*, Directed Acyclic Graph. in the naming rule scheme, the X represents that the formula is conclusion of more than one rule at the same time, what is necesserilly a consequence of a previous collapse.

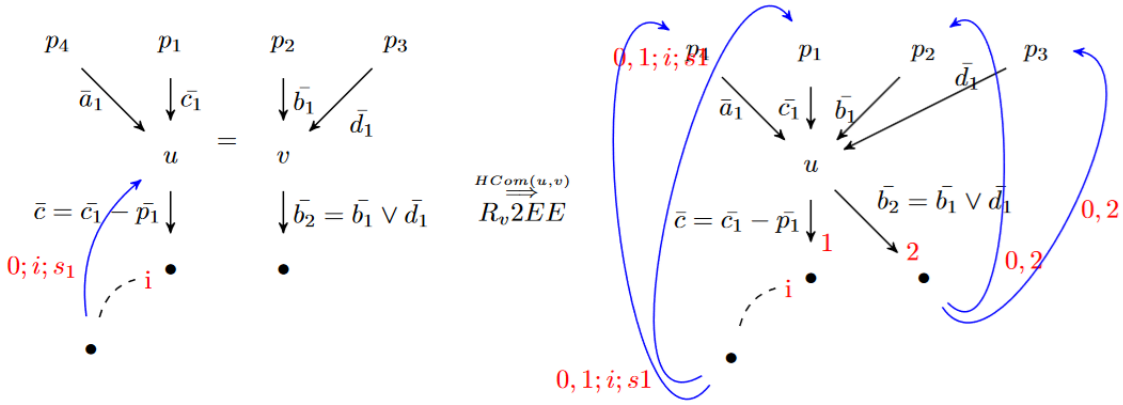

Figure 5.6: **MDE** Compression Rule **R_v2EE**.

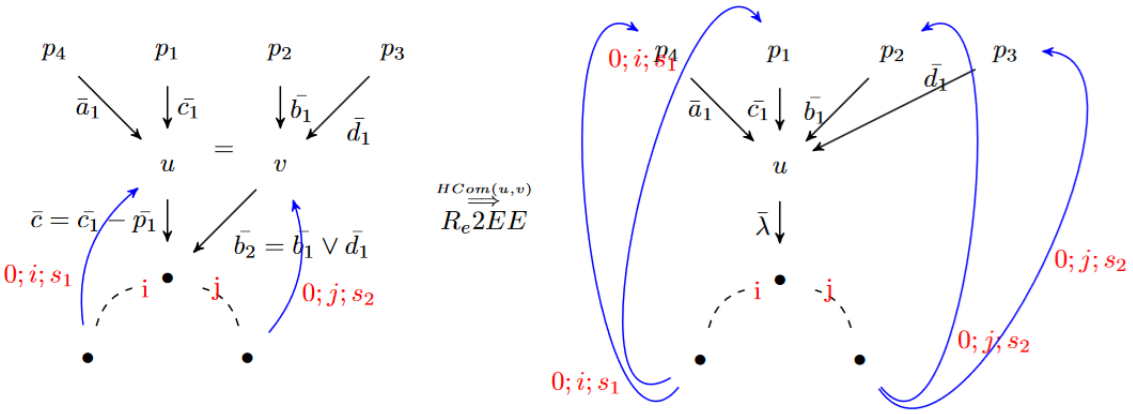

Figure 5.7: **MDE** Compression Rule **R_e2EE**.

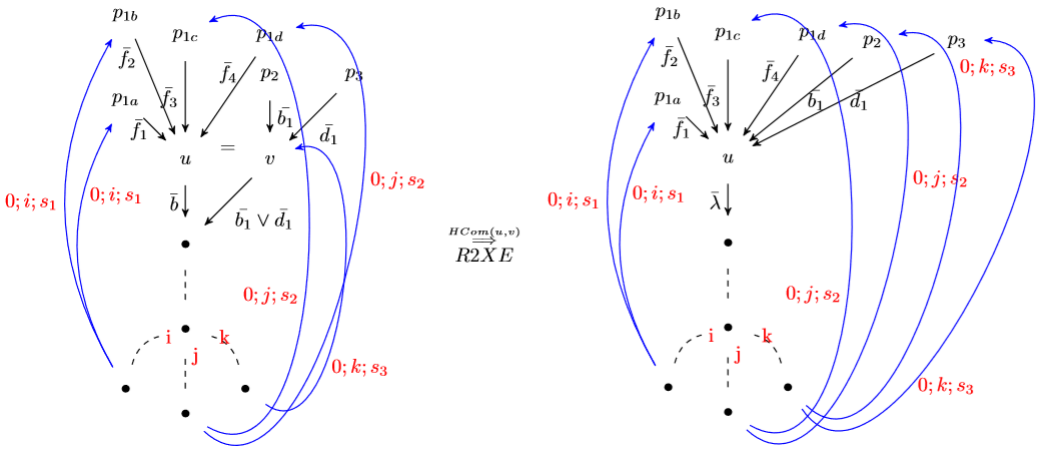

Figure 5.8: **MDE** Compression Rule $\mathbf{R_e2XE}$.

The rule **R0EE** is used to exemplify how to read the pictorial representation of the compression rules of **HC**. The left and the right-hand sides are subgraphs of, respectively, two **DLDS**s, $\mathcal{D}$ and $\mathcal{D}'$. This rule states that one must replace the subgraph represented by the left-hand side by the right-hand side graph in $\mathcal{D}$, resulting in $\mathcal{D}'$, defined below, where $\bullet_a$ is the left $\bullet$ in the figure, while $\bullet_b$ is the right one. In **R0EE** left-hand side, p_i, $i = 1, 3$, u and v are different nodes in the subgraph, such that v and u have the same label(formula), the black arrows are deductive edges, which have as labels the bitstring representing the dependency set denoted by L. For example, $L(\langle p_1, u\rangle) = \bar{c}_1$ shows that the deductive edge $\langle p_1, u\rangle \in E_d^0$ is labeled by the dependency set $Sets(\bar{c}_1)$. The absence of a label on an edge indicates that it is irrelevant to the pattern. A label's node is $\bullet$ whenever it is not relevant what is its label to read the rule. In this case, the $\bullet$ is also used to denote the node. $\bullet$s label different nodes unless explicitly stated by the rule. In figure 5.3, the bullets label different nodes. In the set-theoretical semantics of the rules, $\bullet_a$ and $\bullet_b$ are references to each of the two different nodes [2]. Edges that belong to E_D^i have the colour i; this is the red ordinal[3] number $1, \ldots, n$ on a black deduction edge. The members of E_A, the ancestor edges, are coloured blue, and their labels under P labelling function are red in the picture. For example, $\langle \bullet, p_1\rangle \in E_A$ and $P(\langle \bullet, p_1\rangle) = 1$. Moreover, one has that $\langle u, \bullet\rangle \in E_D^1$ in the graph on the right-hand side of **R0EE**. A **DLDS** $\mathcal{D}$ is specified by a set of nodes V, and multiple sets of deductive edges, $E_D^i \subseteq V \times V$, $i = 0, n$, called coloured edges, and a set of ancestor edges $E_A \subseteq V \times V$, plus some labelling functions. The label λ will be assigned to some edges, i.e., those belonging to E_D^λ. The members of E_D^λ are edges that must have the dependency set calcu-

[3]Note that the formulas themselves can be used as ordinal numbers in this case since they are linearly ordered

lated by the validation verification algorithm that verifies if a **DLDS** is valid [2]. Moreover, a node on the left-hand side must show all the edges,incoming and outgoing. If no edge is drawn, then the node does not appear in the rule.

In a natural deduction derivation, the correct and logical reading starts in the hypothesis and follows down the conclusion analysing the changing of dependency sets obtained by each rule application. The path downwards is the path that links a node to its parent, the latter to its respective parent and so on. In a **dag**, the correct reading is a bit more involving. The result of the application of **R0EE**, in figure 5.3, to the match in figure 5.4 is depicted in figure 5.5. The matching is represented by ellipses around the nodes that are bullets in the rule (**R0EE**) representation instead of the original rectangles. After the application, the matching is still present to us appreciate the effect of the application of **R0EE**. This focus is removed in figure 5.9, showing the derivation resulting from rule **R0EE** application, ready to be used in a new matching by the application of a new rule.

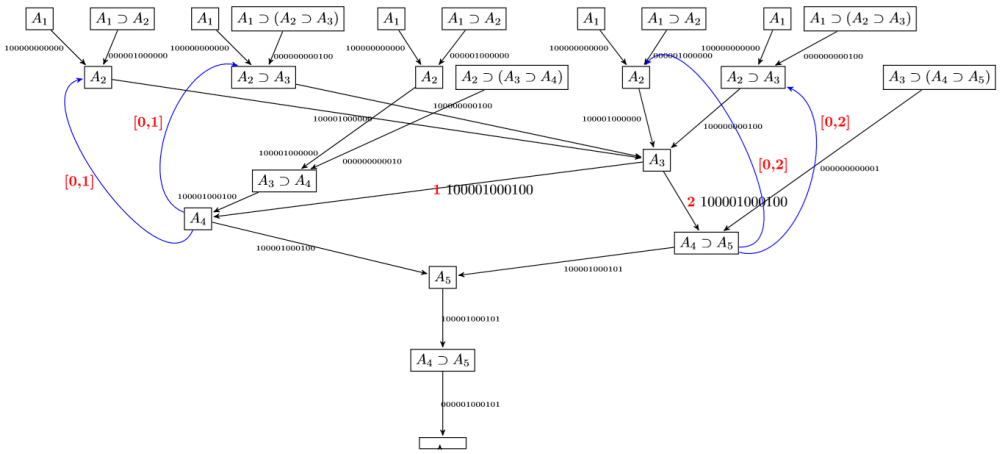

Figure 5.9: Defocused **DLDS** from figure 5.5.

One should observe that the graph in this figure, resulting from a node application collapse, is no longer a tree. It is represented by a **dag**. The collapsed node labelled by the formula A_3 has out-degree 2, i.e., has two outcoming edges labelled with 1 and 2, in red, respectively, besides their respective dependency sets as bitstrings. Labels 1 and 2 inform which path a derivation validation algorithm should follow to have the correct logical dependency from the conclusion to the hypothesis. The path that must be followed from a given node v to obtain the correct reading is given by the label of the ancestor edge incident in v. If there is no ancestor edge incident in a node, then the correct reading is to follow the parenthood path. For example, in figure 5.5, the list $[0, 2]$, in red, labels the edges that go from the node labelled with $A_4 \supset A_5$ to the labelled nodes A_2 and $A_2 \supset A_3$, respectively.

$[0, 2]$ is the path to follow from A_2 to $A_4 \supset A_5$ passing by A_3. The same can be said about the path from $A_2 \supset A_3$ to $A_4 \supset A_5$. These two paths are the only ones present in the tree-like derivation from these two nodes, labelled with A_2 and $A_2 \supset A_3$ to $A_4 \supset A_5$, respectively. Hence, the ancestor edges drive the correct reading of the dag-like derivation regarding the original tree-like derivation before applying the collapsing rules. A reading algorithm reads and validates a dag-like derivation [2]. One should note that the correct paths are preserved after the **R0EE** application, indicating the soundness of the **R0EE**.

The following collapse from the bottom up involves three A_2-labeled nodes in level 4[4]. The first collapse considers the first two occurrences of A_2, from left to right in the dag. The second leftmost A_2 is the minor premise of a $\supset$-Elimination with $A_2 \supset (A_3 \supset A_4)$ as major premise. The rule $\mathbf{R_v2EE}$, in figure 5.6, is applied to the current dag-like deduction in figure 5.9. Figure 5.11 is a graphical rearrangement of this same dag-like derivation, only to have a more evident matching of $\mathbf{R_v2EE}$, in figure 5.10.

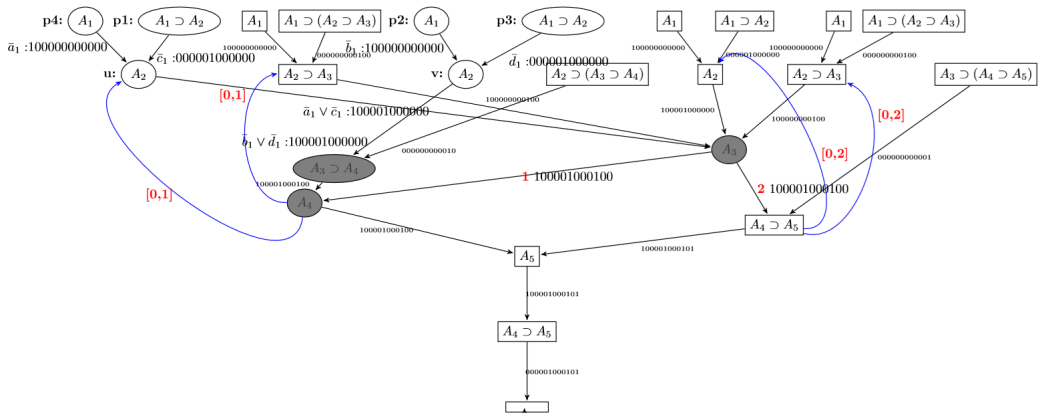

Figure 5.10: Collapsing of nodes u and v: Matching of $\mathbf{R_v2EE}$ to the **DLDS** at figure 5.9.

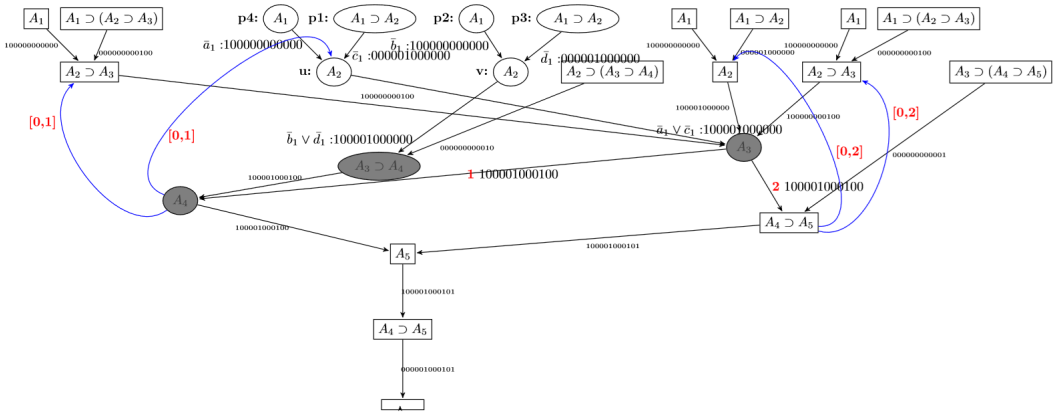

Figure 5.11: Rearrangement of the **DLDS** at figure 5.10.

[4]The **HC** algorithm always deals with levelled dag-like derivations.

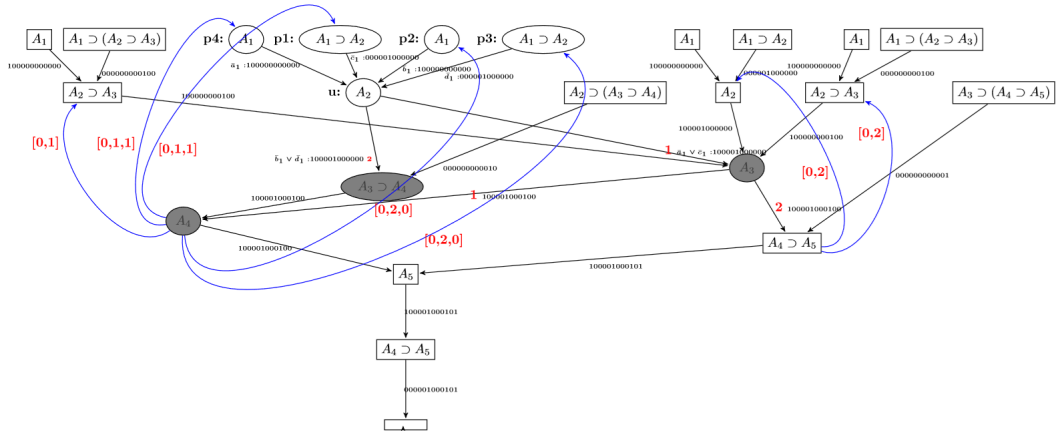

Figure 5.12: Resulting **DLDS** after applying $\mathbf{R_v2EE}$ to the **DLDS** at figure 5.11.

From this point on, the derivation depicted in figure 5.15 is provide by applying the rules **R__e2XE**, shown in figure 5.8, and $\mathbf{R_e2EE}$, shown in figure 5.7, to the **DLDS** at 5.12. The dag-like derivation in figure 5.15 has nodes labelled with the same formula only at the top formulas, i.e., hypothesis or top-formulas. In this case, the four occurrences of A_1 and the three occurrences of $A_1 \supset A_2$. Moreover, it is a simple directed and coloured graph, i.e., for each colour c and pair u and v of nodes, there is at most one edge of colour c from u to v. Suppose that the top formulas are not to be counted. Then, in that case, the obtained dag-like derivation is of size $h.m^3$, where h is the height of the original tree-like natural deduction derivation and m is the number of formulas in the original derivation[5].

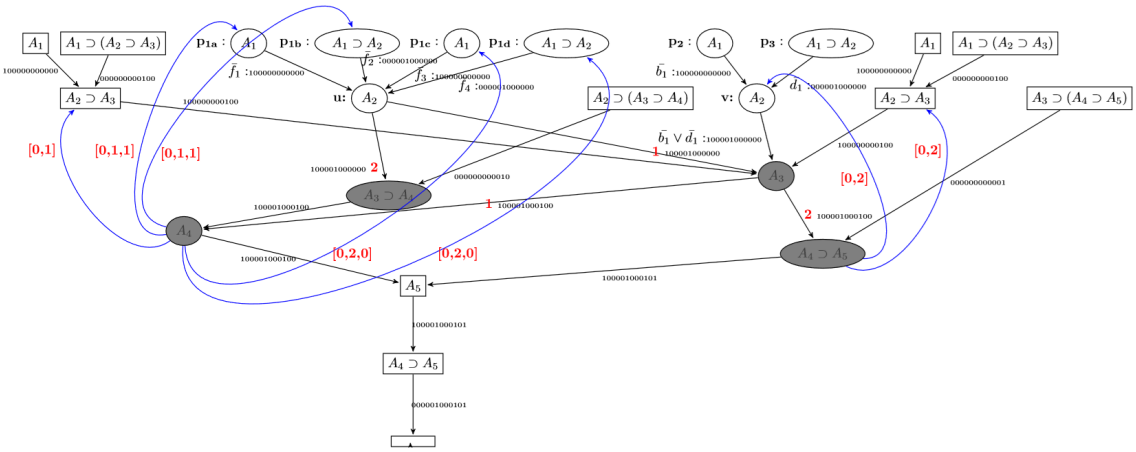

Figure 5.13: Collapsing of nodes u and v: Matching of $\mathbf{R_e2XE}$ to the **DLDS** at 5.12.

[5]Refer to [2] for more detail.

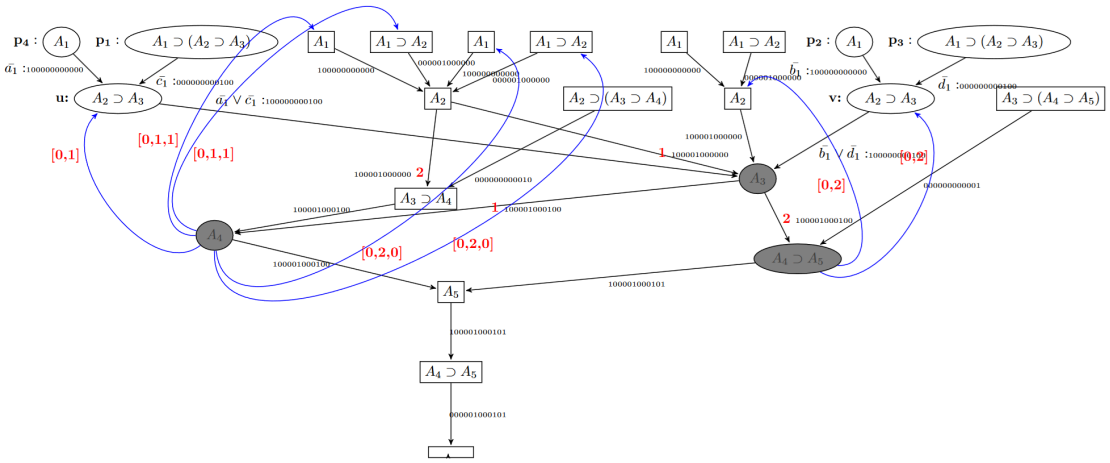

Figure 5.14: Collapsing of nodes u and v: Matching of **R_e2EE** to the **DLDS** at 5.12.

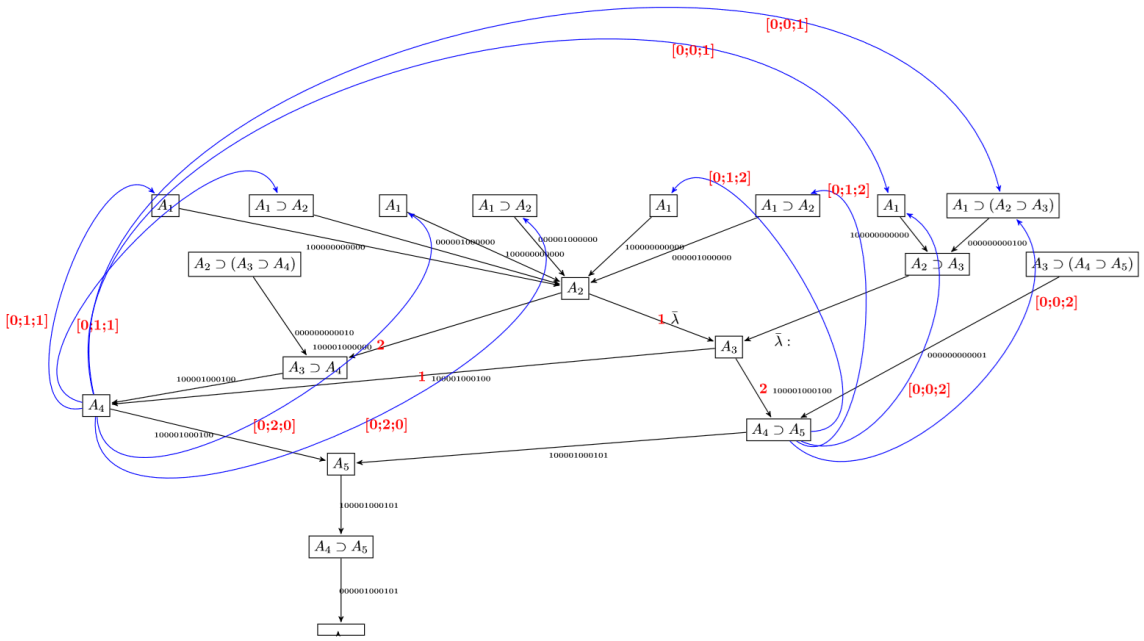

Figure 5.15: Resulting **DLDS** after applying **R_e2XE** and **R_e2EE** to the **DLDS** at 5.12.

The next, and final, step in the compression task is to collapse the top formulas. The **M**oving **D**ownward **E**dges rules (MDE-rules) are used for this task. The application of the MDE-rules moves the ancestor edges down, but preserves the number of ancestor edges, so that there is at most one top formula by level at the end of the compression process, and the same number of edges counted above. One should conclude that the dag-like proof is polynomial on the number of formulas and the height of the tree-like derivation. Do note that some ancestor edges are redundant, and could also be collapsed and eliminated. This happens whenever they connect the same two nodes and have the same

label, meaning they describe the same path between these two nodes. Since this is the case for any two such ancestor edges, then **HC** ends exiting a polynomial **DLDS**[2].

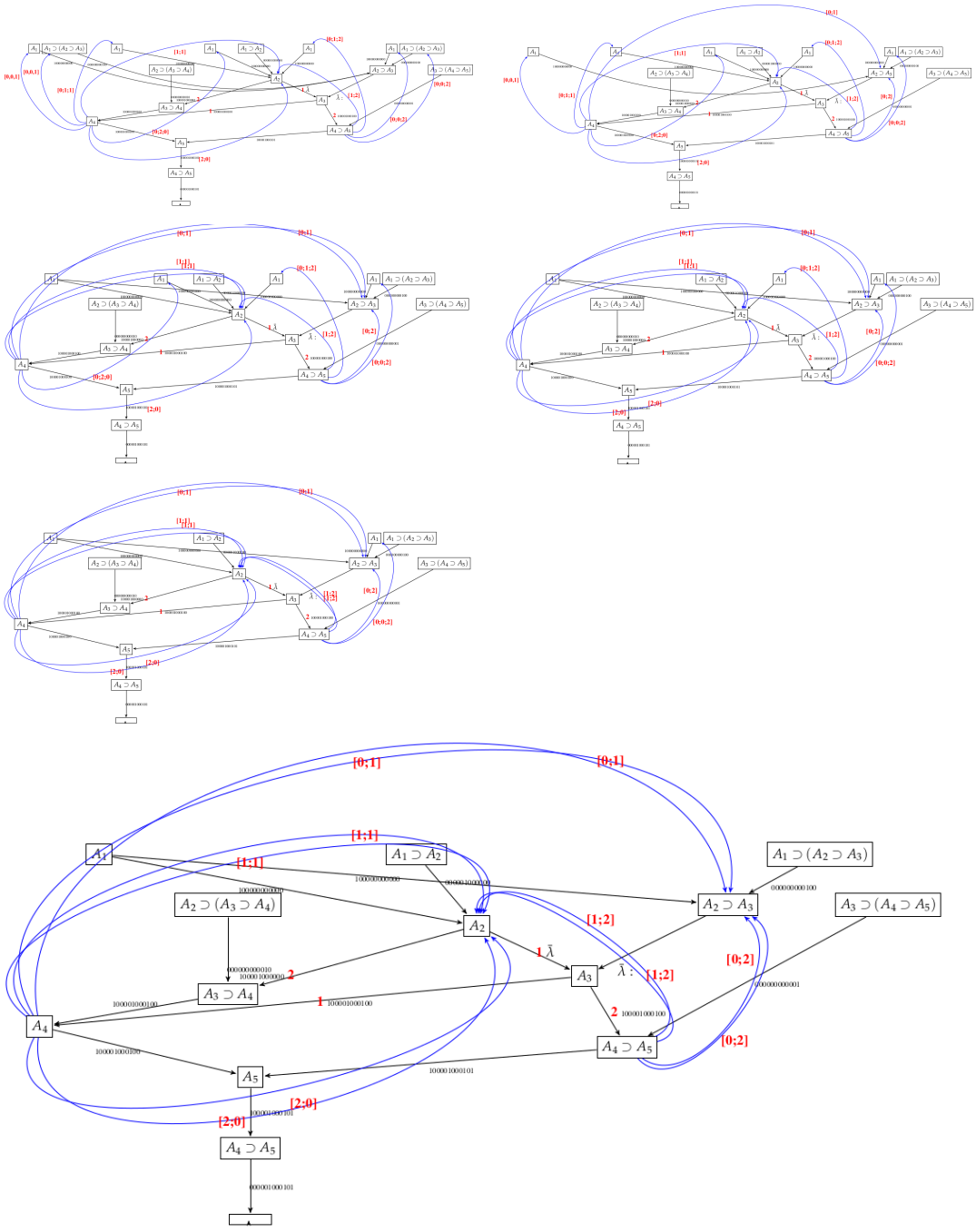

Figure 5.16: Summary of the **MDE** applications to the initial derivation.

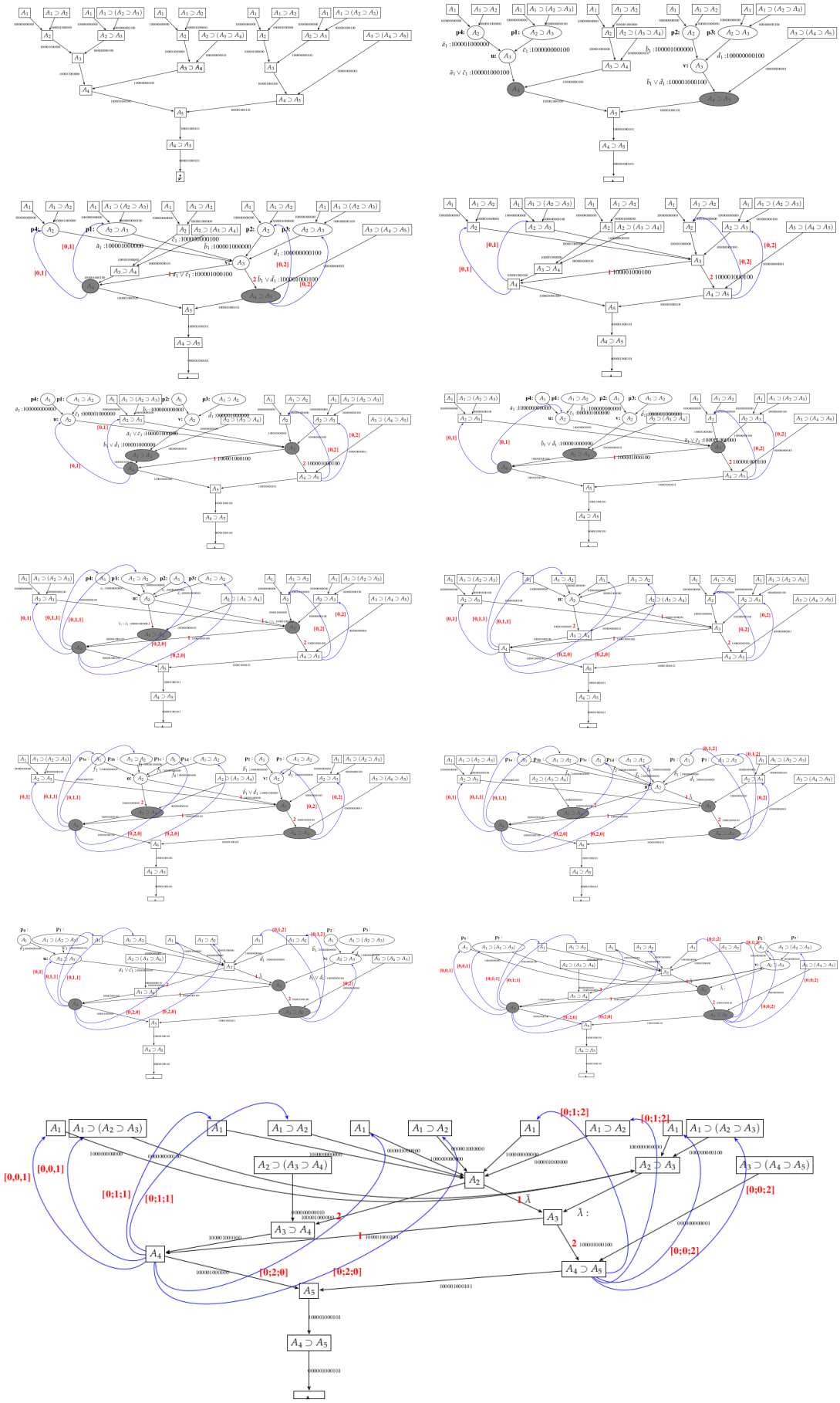

Figure 5.17: Summary of the **MUE** applications to the initial derivation.

6
Formalisation in Lean

This chapter describes the formalization done in Lean (lean4 v0.0.140) for the proof of the following theorem:

Theorem 6.1 (**MUE** Coverage and Soundness of the **HC** Algorithm)**.** *Let* $HCom(u, v)$ *be the application of a* **MUE***-rule nodes* u *and* v *of a* **DLDS** $\mathcal{DLDS}$*. Let* uv *be the resulting collapsed node and* $\mathcal{CLPS}'$ *the resulting* **DLDS** *after the collapse of nodes* u *and* v*. If so, then* uv*, any child-node* u_1 *of* u *and any child-node* v_1 *of* v *all respect the Type Hierarchy of* u *and* v *at* $\mathcal{CLPS}$*.*

6.1
Type Definitions

As the proof is about the compression rules that define $HCom(u, v)$, it is necessary to create a type for the entries u and v. In this context, u and v indicate the nodes to be collapsed by the application of $HCom(u, v)$. However, the information of which nodes u and v represent is insufficient for this proof. Not only the nodes to be collapsed, all neighboring arrows and vertexes around the nodes u and v must be known. The `neighborhood` type is directly defined by the compression rules, and composed of 4 distinct parts: a central node which is used for the collapse (CENTER); a list of deduction edges arriving at that central node (IN); a list of deduction edges exiting from that central node (OUT); and a list of edges of ancestrality pertinent to the compression rule (ANCESTRAL). These parts are given as parameter to the `neighborhood` type constructor. The types of these parameters are the `node`, `deduction`, and `ancestral` types.

```
/- DLDS: Labels -/
inductive Formula where
| atom (SYMBOL : String) : Formula
| implication (ANTECEDENT CONSEQUENT : Formula) : Formula
export Formula (atom implication)
prefix:max "#" => Formula.atom
infixl:66 ">>" => Formula.implication
def Formula.repr (FORMULA : Formula) : String :=
    match FORMULA with
    | (atom SYMBOL) => "#" ++ SYMBOL
    | (implication ANTECEDENT CONSEQUENT) => (Formula.repr ANTECEDENT) ++ ">>" ++
    ↪  (Formula.repr CONSEQUENT)
instance : Repr Formula where reprPrec formula _ := Formula.repr formula
```

```
/- DLDS: Vertices -/
structure Vertex where
node :: (NUMBER : Nat)
        (LEVEL : Nat)
        (FORMULA : Formula)
        (HYPOTHESIS : Bool)
        (COLLAPSED : Bool)
        (PAST : List Nat)   /- Temporary Collapse Metadata -/
deriving Repr
export Vertex (node)
/- DLDS: Deduction Edges -/
structure Deduction where
edge :: (START : Vertex)
        (END : Vertex)
        (COLOUR : Nat)
        (DEPENDENCY : List Formula)
deriving Repr
export Deduction (edge)
/- DLDS: Ancestral Paths -/
structure Ancestral where
path :: (START : Vertex)
        (END : Vertex)
        (COLOURS : List Nat)
deriving Repr
export Ancestral (path)
/- DLDS: Graph -/
structure Graph where
dlds :: (NODES : List Vertex)
        (EDGES : List Deduction)
        (PATHS : List Ancestral)
deriving Repr
export Graph (dlds)
/- DLDS: Neighborhoods -/
structure Neighborhood where
rule :: (CENTER : Vertex)
        (INCOMING : List Deduction)
        (OUTGOING : List Deduction)
        (DIRECT : List Ancestral)
        (INDIRECT : List Ancestral)
deriving Repr
export Neighborhood (rule)
```

Source Code 6.1: Type definitions in Lean for the **D**ag-**L**ike **D**erivability **S**tructure (**DLDS**).

In the `node` type's definition, a node's level and label (an identifier unique to that node) are each represented by a natural number, which must be given as parameters to the type constructor. This representation is justifiable because the number of possible levels/labels in a **DLDS** is always a natural number, so any arbitrary ordering over the set of possible levels/labels creates a bijection between the set of levels/labels and the natural numbers. The level parameter of a node is used to associate nodes of the **DLDS** for collapse, and check if they are at the same level of the **DLDS**. Using a parameter to represent the label of a node makes it possible to differentiate

any two nodes of the tree even when looked at in isolation. On each collapsed node is also kept information about the nodes collapsed into it; permanently when there is no size conflict or temporarily if there is. These parameters allow for a better, more precise categorization of the nodes, something which the proof requires. The last part of a node is its formula, which in this context is a $M_{\supset}$ formula, defined as `formula`. The `deduction` type is used to instantiate a neighborhood's deduction edges while the `ancestral` type is used to instantiate a neighborhood's edges of ancestrality. A deduction edge is composed of: a starting node (START); an end node (END); an identifying colour (COLOUR); and a dependency set (DEPENDENCY). An ancestral-path is composed of: a starting node (START); an end node (END); and an identifying colour path (PATH). The formalisation of both these types is taken as a direct translation from their definition as stated in the previous sections.

6.2 Proving the Main Theorem

Central to the Lean-assisted proof, contained at [1], is the concept of the Type Hierarchy, given as follows:

- ➡ `type0_elimination`, 6.2 which validates if a given neighborhood of the **DLDS** represents a non-collapsed node that is the conclusion of an application of $\supset$-Elimination, as it was in the original tree-like proof, with a non-collapsed parent-node or incoming ancestral-paths;

```
265def type0_elimination (RULE : Neighborhood) : Prop :=
266    ( RULE.CENTER.NUMBER > 0 ) ∧ ( RULE.CENTER.LEVEL > 0 ) ∧ ( RULE.CENTER.HYPOTHESIS = false
       ↪ )
267  ∧ ( RULE.CENTER.COLLAPSED = false ) ∧ ( RULE.CENTER.PAST = [] )
268  ∧ ( ∃(inc_nbr out_nbr : Nat),
269      ∃(antecedent out_fml : Formula),
270      ∃(major_hpt minor_hpt : Bool),
271      ∃(major_dep minor_dep : List Formula),
272      ------------------------------------------------------
273      ( inc_nbr > 0 ) ∧ ( out_nbr > 0 )
274    ∧ RULE.INCOMING = [ edge (node (inc_nbr+1) (RULE.CENTER.LEVEL+1)
       ↪  (antecedent>>RULE.CENTER.FORMULA) major_hpt false []) /- Left Child & Major Premise
       ↪  -/
275                            RULE.CENTER
276                            0
277                            #major_dep,
278                       edge (node inc_nbr (RULE.CENTER.LEVEL+1) antecedent minor_hpt false
                          ↪  []) /- Right Child & Minor Premise -/
279                            RULE.CENTER
280                            0
281                            #minor_dep ]
282    ∧ RULE.OUTGOING = [ edge RULE.CENTER
283                            (node out_nbr (RULE.CENTER.LEVEL-1) out_fml false false [])
284                            0
```

```
285                             (minor_dep ∪ major_dep) ]
286     ∧ RULE.DIRECT   = []
287     ∧ RULE.INDIRECT = [] )
```

Source Code 6.2: Type-0 Elimination method definition in Lean.

➡ `type0_introduction`, 6.3 which validates if a given neighborhood of the **DLDS** represents a non-collapsed node that is the conclusion of an application of ⊃-Introduction, as it was in the original tree-like proof, with a non-collapsed parent-node or incoming ancestral-paths;

```
290 def type0_introduction (RULE : Neighborhood) : Prop :=
291     ( RULE.CENTER.NUMBER > 0 ) ∧ ( RULE.CENTER.LEVEL > 0 ) ∧ ( RULE.CENTER.HYPOTHESIS = false
        ↪ )
292   ∧ ( RULE.CENTER.COLLAPSED = false ) ∧ ( RULE.CENTER.PAST = [] )
293   ∧ ( ∃(inc_nbr out_nbr : Nat),
294       ∃(antecedent consequent out_fml : Formula),
295       ∃(inc_dep : List Formula),
296     ------------------------------------------------------
297       ( RULE.CENTER.FORMULA = antecedent>>consequent )
298     ∧ ( inc_nbr > 0 ) ∧ ( out_nbr > 0 )
299     ∧ RULE.INCOMING = [ edge (node inc_nbr (RULE.CENTER.LEVEL+1) consequent false false [])
        ↪  /- Unique Child & Sole Premise -/
300                             RULE.CENTER
301                             0
302                             #inc_dep ]
303     ∧ RULE.OUTGOING = [ edge RULE.CENTER
304                             (node out_nbr (RULE.CENTER.LEVEL-1) out_fml false false [])
305                             0
306                             (inc_dep - [antecedent]) ]
307     ∧ RULE.DIRECT   = []
308     ∧ RULE.INDIRECT = [] )
```

Source Code 6.3: Type-0 Introduction method definition in Lean.

➡ `type0_hypothesis`, 6.4 which validates if a given neighborhood of the **DLDS** represents a non-collapsed node that is a hypothesis, as it was in the original tree-like proof, with a non-collapsed parent-node or incoming ancestral-paths;

```
311 def type0_hypothesis (RULE : Neighborhood) : Prop :=
312     ( RULE.CENTER.NUMBER > 0 ) ∧ ( RULE.CENTER.LEVEL > 0 ) ∧ ( RULE.CENTER.HYPOTHESIS = true
        ↪ )
313   ∧ ( RULE.CENTER.COLLAPSED = false ) ∧ ( RULE.CENTER.PAST = [] )
314   ∧ ( ∃(out_nbr : Nat),
315       ∃(out_fml : Formula),
316     ------------------------------------------------------
317       ( out_nbr > 0 )
318     ∧ RULE.INCOMING = []
319     ∧ RULE.OUTGOING = [ edge RULE.CENTER
320                             (node out_nbr (RULE.CENTER.LEVEL-1) out_fml false false [])
321                             0
322                             [RULE.CENTER.FORMULA] ]
323     ∧ RULE.DIRECT   = []
324     ∧ RULE.INDIRECT = [] )
```

Source Code 6.4: Type-0 Hypothesis method definition in Lean.

➡ `type1_pre_collapse`, 6.5 which validates if a given neighborhood of the **DLDS** represents a pre-collapsed node that is the preview of the future collapse of a type-0 node;

➡ `type1_collapse`, 6.6 which validates if a given neighborhood of the **DLDS** represents a collapsed node resulting from the collapse: of a type-0 and a type-0; or a type-1 and a type-0 nodes;

```
def type1_pre_collapse (RULE : Neighborhood) : Prop :=
    /- Check Center -/-------------------------------------------------------------------
    ( ( RULE.CENTER.NUMBER > 0 ) ∧ ( RULE.CENTER.LEVEL > 0 )
    ∧ ( RULE.CENTER.COLLAPSED = false )
    ∧ ( RULE.CENTER.PAST = [] )
    /- Check Deduction Edges -/----------------------------------------------------------
    ∧ ( ( RULE.INCOMING = [] ) ↔ ( RULE.CENTER.HYPOTHESIS = true ) )
    ∧ ( List.length (RULE.INCOMING) ≤ 2 )
    ∧ ( ∃(out : Deduction), ( RULE.OUTGOING = [out] ) )
    ∧ ( ∀{OUT₁ OUT₂ : Deduction}, ( OUT₁ ∈ RULE.OUTGOING ) →
                                  ( OUT₂ ∈ RULE.OUTGOING ) →
                                  ( OUT₁.COLOUR > 0 ∨ OUT₂.COLOUR > 0 ) →
                                  ( ( OUT₁.COLOUR = OUT₂.COLOUR ) ↔ ( OUT₁ = OUT₂ ) ) )
    /- Check Ancestral Paths -/----------------------------------------------------------
    ∧ ( RULE.DIRECT = [] )
    ∧ ( ∀{ind₁ ind₂ : Ancestral}, ( ind₁ ∈ RULE.INDIRECT ) →
                                  ( ind₂ ∈ RULE.INDIRECT ) → ( ( ind₁.COLOURS = ind₂.COLOURS
                                  ↪  ) ↔ ( ind₁.START = ind₂.START ) ) )
    ∧ ( List.length (RULE.INDIRECT) = List.length (RULE.INCOMING) )
    ∧ ( ∀{ind : Ancestral}, ( ind ∈ RULE.INDIRECT ) → ( ind.COLOURS = [0,
    ↪  RULE.CENTER.NUMBER] ) )
    /- Generic Properties -/-------------------------------------------------------------
    ∧ ( type_incoming RULE ) ∧ ( type_outgoing₁ RULE )
    ∧ ( type_indirect RULE ) )
```

Source Code 6.5: Type-1 Pre-Collapse method definition in Lean.

```
def type1_collapse (RULE : Neighborhood) : Prop :=
    /- Check Center -/-------------------------------------------------------------------
    ( ( RULE.CENTER.NUMBER > 0 ) ∧ ( RULE.CENTER.LEVEL > 0 )
    ∧ ( RULE.CENTER.COLLAPSED = true )
    ∧ ( ∃(past : Nat)(pasts : List Nat), ( check_numbers (past::pasts) )
                                       ∧ ( RULE.CENTER.PAST = (past::pasts) ) )
    /- Check Deduction Edges -/----------------------------------------------------------
    ∧ ( ( RULE.INCOMING = [] ) → ( RULE.CENTER.HYPOTHESIS = true ) )
    ∧ ( ∃(out : Deduction)(outs : List Deduction), ( RULE.OUTGOING = (out::outs) ) )
    ∧ ( ∀{OUT₁ OUT₂ : Deduction}, ( OUT₁ ∈ RULE.OUTGOING ) →
                                  ( OUT₂ ∈ RULE.OUTGOING ) →
                                  ( OUT₁.COLOUR > 0 ∨ OUT₂.COLOUR > 0 ) →
                                  ( ( OUT₁.COLOUR = OUT₂.COLOUR ) ↔ ( OUT₁ = OUT₂ ) ) )
    /- Check Ancestral Paths -/----------------------------------------------------------
    ∧ ( RULE.DIRECT = [] )
    ∧ ( List.length (RULE.INDIRECT) = List.length (RULE.INCOMING) )
```

```
572     ∧ ( ∀{ind : Ancestral}, ( ind ∈ RULE.INDIRECT ) → ( ∃(colour : Nat), ( ind.COLOURS = [0,
        ↪  colour] ) ) )
573     /- Generic Properties -/----------------------------------------------------------------
574     ∧ ( type_incoming RULE ) ∧ ( type_outgoing₁ RULE )
575     ∧ ( type_indirect RULE ) )
```

Source Code 6.6: Type-1 Collapse method definition in Lean.

➡ `type2_elimination`, 6.7 which validates if a given neighborhood of the **DLDS** represents a non-collapsed node that is the conclusion of an application of ⊃-Elimination, but not as it was in the original tree-like proof, with a collapsed parent-node and a single incoming ancestral-path;

```
328def type2_elimination (RULE : Neighborhood) : Prop :=
329     ( RULE.CENTER.NUMBER > 0 ) ∧ ( RULE.CENTER.LEVEL > 0 ) ∧ ( RULE.CENTER.HYPOTHESIS = false
        ↪  )
330   ∧ ( RULE.CENTER.COLLAPSED = false ) ∧ ( RULE.CENTER.PAST = [] )
331   ∧ ( ∃(inc_nbr out_nbr anc_nbr anc_lvl : Nat),
332       ∃(antecedent out_fml anc_fml : Formula),
333       ∃(major_hpt minor_hpt out_hpt : Bool),
334       ∃(major_dep minor_dep : List Formula),
335       ∃(past colour : Nat)(pasts colours : List Nat),
336       ------------------------------------------------------
337       ( inc_nbr > 0 ) ∧ ( out_nbr > 0 )
338     ∧ ( anc_nbr > 0 ) ∧ ( anc_lvl + List.length (0::colour::colours) = RULE.CENTER.LEVEL )
339     ∧ ( colour ∈ (out_nbr::past::pasts) ) ∧ ( check_numbers (past::pasts) ) ∧ (
        ↪  check_numbers (colour::colours) )
340     ∧ RULE.INCOMING = [ edge (node (inc_nbr+1) (RULE.CENTER.LEVEL+1)
        ↪  (antecedent>>RULE.CENTER.FORMULA) major_hpt false []) /- Right Child & Major Premise
        ↪  -/
341                               RULE.CENTER
342                               0
343                               #major_dep,
344                         edge (node inc_nbr (RULE.CENTER.LEVEL+1) antecedent minor_hpt false
                            ↪  [])                              /- Left Child & Minor Premise -/
345                               RULE.CENTER
346                               0
347                               #minor_dep ]
348     ∧ RULE.OUTGOING = [ edge RULE.CENTER
349                               (node out_nbr (RULE.CENTER.LEVEL-1) out_fml out_hpt true
                                  ↪  (past::pasts))
350                               0
351                               (minor_dep ∪ major_dep) ]
352     ∧ RULE.DIRECT   = [ path (node anc_nbr anc_lvl anc_fml false false [])
353                               RULE.CENTER
354                               (0::colour::colours) ]
355     ∧ RULE.INDIRECT = [] )
```

Source Code 6.7: Type-2 Elimination method definition in Lean.

➡ `type2_introduction`, 6.8 which validates if a given neighborhood of the **DLDS** represents a non-collapsed node that is the conclusion of an application of ⊃-Introduction, but not as it was in the original tree-like proof, with a collapsed parent-node and a single incoming ancestral-path;

```
def type2_introduction (RULE : Neighborhood) : Prop :=
  ( RULE.CENTER.NUMBER > 0 ) ∧ ( RULE.CENTER.LEVEL > 0 ) ∧ ( RULE.CENTER.HYPOTHESIS = false )
∧ ( RULE.CENTER.COLLAPSED = false ) ∧ ( RULE.CENTER.PAST = [] )
∧ ( ∃(inc_nbr out_nbr anc_nbr anc_lvl : Nat),
    ∃(antecedent consequent out_fml anc_fml : Formula),
    ∃(out_hpt : Bool),
    ∃(inc_dep : List Formula),
    ∃(past colour : Nat)(pasts colours : List Nat),
  ------------------------------------------------------
    ( RULE.CENTER.FORMULA = antecedent>>consequent )
  ∧ ( inc_nbr > 0 ) ∧ ( out_nbr > 0 )
  ∧ ( anc_nbr > 0 ) ∧ ( anc_lvl + List.length (0::colour::colours) = RULE.CENTER.LEVEL )
  ∧ ( colour ∈ (out_nbr::past::pasts) ) ∧ ( check_numbers (past::pasts) ) ∧ ( check_numbers (colour::colours) )
  ∧ RULE.INCOMING = [ edge (node inc_nbr (RULE.CENTER.LEVEL+1) consequent false false [])  /- Unique Child & Sole Premise -/
                           RULE.CENTER
                           0
                           #inc_dep ]
  ∧ RULE.OUTGOING = [ edge RULE.CENTER
                           (node out_nbr (RULE.CENTER.LEVEL-1) out_fml out_hpt true (past::pasts))
                           0
                           (inc_dep - [antecedent]) ]
  ∧ RULE.DIRECT   = [ path (node anc_nbr anc_lvl anc_fml false false [])
                           RULE.CENTER
                           (0::colour::colours) ]
  ∧ RULE.INDIRECT = [] )
```

Source Code 6.8: Type-2 Introduction method definition in Lean.

➡ `type2_hypothesis`, 6.9 which validates if a given neighborhood of the **DLDS** represents a non-collapsed node that is a hypothesis, but not as it was in the original tree-like proof, with a collapsed parent-node and a single incoming ancestral-path;

```
def type2_hypothesis (RULE : Neighborhood) : Prop :=
  ( RULE.CENTER.NUMBER > 0 ) ∧ ( RULE.CENTER.LEVEL > 0 ) ∧ ( RULE.CENTER.HYPOTHESIS = true )
∧ ( RULE.CENTER.COLLAPSED = false ) ∧ ( RULE.CENTER.PAST = [] )
∧ ( ∃(out_nbr anc_nbr anc_lvl : Nat),
    ∃(out_fml anc_fml : Formula),
    ∃(out_hpt : Bool),
    ∃(past colour : Nat)(pasts colours : List Nat),
  ------------------------------------------------------
    ( out_nbr > 0 )
  ∧ ( anc_nbr > 0 ) ∧ ( anc_lvl + List.length (0::colour::colours) = RULE.CENTER.LEVEL )
  ∧ ( colour ∈ (out_nbr::past::pasts) ) ∧ ( check_numbers (past::pasts) ) ∧ ( check_numbers (colour::colours) )
  ∧ RULE.INCOMING = []
  ∧ RULE.OUTGOING = [ edge RULE.CENTER
                           (node out_nbr (RULE.CENTER.LEVEL-1) out_fml out_hpt true (past::pasts))
```

```
                            0
                            [RULE.CENTER.FORMULA] ]
    ∧ RULE.DIRECT   = [ path (node anc_nbr anc_lvl anc_fml false false [])
                            RULE.CENTER
                            (0::colour::colours) ]
    ∧ RULE.INDIRECT = [] )
```

Source Code 6.9: Type-2 Hypothesis method definition in Lean.

- ➡ `type3_pre_collapse`, 6.10 which validates if a given neighborhood of the **DLDS** represents a pre-collapsed node that is the preview of the future collapse of a type-2 node;
- ➡ `type3_collapse`, 6.11 which validates if a given neighborhood of the **DLDS** represents a collapsed node resulting from the collapse of: a type-0 and a type-0; or a type-0 and a type-2; or a type-2 and a type-0; or a type-2 and a type-2; or a type-1 and a type-0; or a type-1 and a type-2; or a type-3 and a type-0; or a type-3 and a type-2 nodes.

```
def type3_pre_collapse (RULE : Neighborhood) : Prop :=
    /- Check Center -/-------------------------------------------------------------------
    ( ( RULE.CENTER.NUMBER > 0 ) ∧ ( RULE.CENTER.LEVEL > 0 )
    ∧ ( RULE.CENTER.COLLAPSED = false )
    ∧ ( RULE.CENTER.PAST = [] )
    /- Check Deduction Edges -/----------------------------------------------------------
    ∧ ( ( RULE.INCOMING = [] ) ↔ ( RULE.CENTER.HYPOTHESIS = true ) )
    ∧ ( List.length (RULE.INCOMING) ≤ 2 )
    ∧ ( ∃(out : Deduction), ( RULE.OUTGOING = [out] ) )
    ∧ ( ∀{OUT₁ OUT₂ : Deduction}, ( OUT₁ ∈ RULE.OUTGOING ) →
                                  ( OUT₂ ∈ RULE.OUTGOING ) →
                                  ( OUT₁.COLOUR > 0 ∨ OUT₂.COLOUR > 0 ) →
                                  ( ( OUT₁.COLOUR = OUT₂.COLOUR ) ↔ ( OUT₁ = OUT₂ ) ) )
    /- Check Ancestral Paths -/----------------------------------------------------------
    ∧ ( ( RULE.CENTER.HYPOTHESIS = false ) → ( RULE.DIRECT = [] ) )
    ∧ ( ( RULE.DIRECT ≠ [] ) → ( RULE.CENTER.HYPOTHESIS = true ) )
    ∧ ( ( RULE.DIRECT = [] ) ∨ ( ∃(dir : Ancestral), ( RULE.DIRECT = [dir] ) ) )
    ∧ ( ∀{ind₁ ind₂ : Ancestral}, ( ind₁ ∈ RULE.INDIRECT ) →
                                  ( ind₂ ∈ RULE.INDIRECT ) → ( ( ind₁.COLOURS = ind₂.COLOURS
                                  ↪  ) ↔ ( ind₁.START = ind₂.START ) ) )
    ∧ ( List.length (RULE.INDIRECT) = List.length (RULE.INCOMING) )
    /- Generic Properties -/-------------------------------------------------------------
    ∧ ( type_incoming RULE ) ∧ ( type_outgoing₃ RULE )
    ∧ ( type_direct RULE ) ∧ ( type_indirect RULE ) )
```

Source Code 6.10: Type-3 Pre-Collapse method definition in Lean.

```
def type3_collapse (RULE : Neighborhood) : Prop :=
    /- Check Center -/-------------------------------------------------------------------
    ( ( RULE.CENTER.NUMBER > 0 ) ∧ ( RULE.CENTER.LEVEL > 0 )
    ∧ ( RULE.CENTER.COLLAPSED = true )
    ∧ ( ∃(past : Nat)(pasts : List Nat), ( check_numbers (past::pasts) )
                                       ∧ ( RULE.CENTER.PAST = (past::pasts) ) )
```

```
610    /- Check Deduction Edges -/-------------------------------------------------------------
611    ∧ ( ( RULE.INCOMING = [] ) → ( RULE.CENTER.HYPOTHESIS = true ) )
612    ∧ ( ∃(out : Deduction)(outs : List Deduction), ( RULE.OUTGOING = (out::outs) ) )
613    ∧ ( ∀{OUT₁ OUT₂ : Deduction}, ( OUT₁ ∈ RULE.OUTGOING ) →
614                                  ( OUT₂ ∈ RULE.OUTGOING ) →
615                                  ( OUT₁.COLOUR > 0 ∨ OUT₂.COLOUR > 0 ) →
616                                  ( ( OUT₁.COLOUR = OUT₂.COLOUR ) ↔ ( OUT₁ = OUT₂ ) ) )
617    /- Check Ancestral Paths -/-------------------------------------------------------------
618    ∧ ( ( RULE.CENTER.HYPOTHESIS = false ) → ( RULE.DIRECT = [] ) )
619    ∧ ( ( RULE.DIRECT ≠ [] ) → ( RULE.CENTER.HYPOTHESIS = true ) )
620    ∧ ( List.length (RULE.INDIRECT) = List.length (RULE.INCOMING) )
621    /- Generic Properties -/----------------------------------------------------------------
622    ∧ ( type_incoming RULE ) ∧ ( type_outgoing₃ RULE )
623    ∧ ( type_direct RULE ) ∧ ( type_indirect RULE ) )
```

Source Code 6.11: Type-3 Collapse method definition in Lean.

As a consequence of the above, if the subset of **MUE**-rules of **HC** is applied to a valid **DLDS**, then one can be certain that it eventually halts exiting a **DLDS** that has no level with two nodes labelled with the same formula, aside from its top-formulas. From the Main Theorem 6.1, take that every matching involving a collapsed node resulting from the application of a type-0 or type-1 rule will be mapped by a type-1 rule. Similarly, every matching involving a collapsed node resulting from the application of a type-0, type-1, type-2 or type-3 rule will be mapped by a type-3 rule. The inductive part of the proof, which states that any child-node of a collapsed node also obeys the Type Hierarchy, similarly states that non-collapsed nodes go from being type-0 to being type-2 nodes after the collapse of their parent-node. So long as there is a **MUE**-rule among the compression rules that maps the matching nodes, **HC** will collapse them. Therefore, **HC** will not halt before collapsing every node aside for the top-formulas.

- `check_collapse_nodes`, [1], which checks if the neighborhoods of two given nodes are valid targets for the collapse of those nodes;
- `check_collapse_edges`, [1], which checks if the neighborhoods of two given nodes are valid targets for the collapse of those nodes and the one outgoing edge those nodes have in common;
- `pre_collapse`, [1], which takes one applicable neighborhood and exits a pre-collapsed neighborhood;
- `collapse`, [1], which takes two applicable neighborhoods and exits a collapsed neighborhood; and
- `Graph.is_collapse`, [1], which indicates the new **DLDS** after the collapse of two given nodes.

7
Future Work

With the intent of expanding the work presented in this article, based on the conclusions and on the insights taken in the course of its making, some focuses in regards to the future developments of this work should be proposed. First and foremost, it is my intention to continue writing the Lean-assisted proof, now including the coverage and soundness of the **MDE** subset of compression rules. After this, the next step is to formalise the result that the final, fully-compressed **DLDS** resulting of an application of **HC**, up to and including all top-formulas, preserves the validity of the original tree-like derivation, via the preservation of **Flow**, as alluded to in **Theorem 11** in [2]. This will, in all regards, finish a complete Lean-assisted proof for the coverage and soundness of **HC**.

Aside from this more natural continuation, perhaps even to be done in parallel to it, is the exploration of a formalisation about other properties of **HC**, besides coverage and soundness, such as the proof of the compression rate. Work in this regard should add to the discussion about $CoNP = NP$, in favor and corroborating the result. Another possible development would be to look into other viable and relevant applications of **HC**. These such applications, like for the proofs of non-Hamiltonicity for non-Hamiltonian graphs, shown in [11], could yield interesting insights.

Lastly, with the current formalisation as a basis, fully-implementing **HC** in Lean is also an option. To that effect, such an implementation would make the Lean-assisted proof much more accessible in and of itself. A reader still skeptical of the Lean-assisted proof, could be more easily persuaded of its semantical significance by following the execution of **HC** with a concrete example, as defined by the formalisation.

8
Conclusion

The goal of this work is to convince the reader that **HC** halts for every $M_{\supset}$ tautology, exiting a valid **DLDS** with no two equal nodes on the same level, aside from the top-formulas. To that end, a proof of the coverage and soundness of the **MUE** subset of compression rules of **HC** was shown.

During the formalisation in Lean, finding succinct enough concepts and data-types to define a **DLDS** was a particular challenge, especially on the early stages. An argument could be made that this characteristic is due to the nature of the tree type, of which a **DLDS** is based, which is not at all intuitive when defining the **DLDS** as a graph in Lean. A **DLDS**, similarly to a tree-like proof, has levels over which recursion and induction can be applied. This, though extremely important for some aspects of the proof, is also not at all needed to prove other parts of the result. Thus, defining an adjacent type, the `neighborhood` type, allowed me to work with a type which has a much more direct definition based on the compression rules, instead of a **DLDS** type, when required.

One thing that also became apparent along the process of formalisation, was how difficult it is to sometimes formalize succinct, sometimes even simple concepts and definitions regarding some data-types, like trees and graphs, and processes, like the derivation rules of $M_{\supset}$. For example, the fact that the set of nodes in a connected graph has a strong correlation to its set of edges, or that the formula labelling a node of a dag-like derivation is restricted in relation to the formulas labelling its own node's child and parent node(s).

It is also worth repeating that the assisted proof did not end up as a 1-to-1 translation of the proofs shown in [2]. Something similar also happened during the previous work at [8], to the point that, after also looking into other works done in the field of formal verification, it would seem that this is a common occurrence. An assisted proof, by all means, must be capable of standing on its own merits. Neither it needs nor it benefits from being the perfect copy of another proof, which is what any pen-and-paper proof of its same result would be: another proof. Formal verification and interactive/assisted theorem proving present their own advantages and disadvantages in relation to more traditional proofs, and it is perfectly normal for different means to yield different products.

Bibliography

[1] FILHO, R.C.M.B., **Horizontal-Compression**: A lean-assisted proofs about properties of the hc algorithm (2023).
URL https://github.com/Robilsu/Horizontal-Compression

[2] HAEUSLER, E.H.; JÚNIOR, J.F.C.B.; FILHO, R.C.M.B., **On the horizontal compression of dag-derivations in minimal purely implicational logic** (arXiv:2206.02300,2023).
URL https://arxiv.org/abs/2206.02300

[3] PRAWITZ, D., **Natural Deduction: Proof-Theoretical Study**, Dover Books on Mathematics, Dover Publications, 2006.
URL https://books.google.com.br/books?id=sJj3DQAAQBAJ

[4] AVIGAD, J.; DE MOURA, L.; KONG, S.; ULLRICH; S., **Theorem Proving in Lean 4** (2022).
URL https://lean-lang.org/theorem_proving_in_lean4/

[5] AVIGAD, J.; DE MOURA, L.; KONG, S.; ULLRICH; S., **Lean Manual** (2022).
URL https://lean-lang.org/lean4/doc/

[6] FILHO, R.C.M.B.; HAEUSLER, E.H.; SANTOS, J.B., **Towards a proof in Lean about the Horizontal Compression of Dag-Like Derivations in Minimal Purely Implicational Logic**, in: The Seventeenth International Workshop on Logical and Semantic Frameworks, with Applications, Belo Horizonte, Minas Gerais, Brazil, 2022, pp. 8–23.
URL https://lsfa2022.github.io/lsfa2022-preproc.pdf#page=11

[7] FILHO, R.C.M.B.; HAEUSLER, E.H.; SANTOS, J.B., **On the Coverage Property of a Derivation Compression Algorithm**, in: Anais do IV Workshop Brasileiro de Lógica, SBC, Porto Alegre, RS, Brasil, 2023, pp. 1–8. doi:10.5753/wbl.2023.230566.
URL https://sol.sbc.org.br/index.php/wbl/article/view/25167

[8] FILHO, R.C.M.B., **Theorem-of-Branch-Redundancy**: A formalization in lean showing that every exponentially big proof in minimal im-

plicational logic is also exponentially redundant (2019).
URL https://github.com/Robilsu/Theorem-of-Branch-Redundancy

[9] GORDEEV, L.; HAEUSLER, E.H., **Proof Compression and NP Versus PSPACE**, Studia Logica 107 (1) (2019) 53–83. doi:10.1007/s11225-017-9773-5.
URL https://doi.org/10.1007/s11225-017-9773-5

[10] GORDEEV, L.; HAEUSLER, E.H., **Proof Compression and NP vs PSPACE II**, Bulletin of the Section of Logic 49 (3) (2020) 213–230. doi:10.18778/0138-0680.2020.16.
URL https://doi.org/10.18778/0138-0680.2020.16

[11] GORDEEV, L.; HAEUSLER, E.H., **Proof Compression and NP vs PSPACE II: Addendum**, Bulletin of the Section of Logic 51 (2) (2022) 197–205. doi:10.18778/0138-0680.2022.01.
URL https://doi.org/10.18778/0138-0680.2022.01

[12] HUDELMAIER, J., **An O(n log n)-Space Decision Procedure for Intuitionistic Propositional Logic**, Journal of Logic and Computation 3 (1) (1993) 63–75. doi:10.1093/logcom/3.1.63.
URL https://doi.org/10.1093/logcom/3.1.63

[13] JÚNIOR, J.F.C.B.; HAEUSLER, E.H., **A comparative study on compression techniques for Propositional Proofs**, in: Book of Abstracts, 19th Braz. Meeting on Logic, João Pessoa, Paraíba, Brazil, 2019, pp. 85–86.
URL https://ebl2019.ci.ufpb.br/assets/Book_of_Abstracts_EBL_2019.pdf#page=86